Advanced FPGA design

ISOTDAQ 2016 @ Rehovot (Israel)

29/01/2016

Manoel Barros Marin

BE-BI-QP

Advanced FPGA design

ISOTDAQ 2016 @ Rehovot (Israel)

29/01/2016

Outline:

- … from the previous lesson
- Key concepts about FPGA design
- FPGA gateware (firmware) design work flow
- Summary

Manoel Barros Marin

BE-BI-QP

Advanced FPGA design

ISOTDAQ 2016 @ Rehovot (Israel)

29/01/2016

Outline:

- **… from the previous lesson**
  - Key concepts about HDL
  - FPGA gateware (firmware) design work flow
  - Summary

Manoel Barros Marin

BE-BI-QP

… from the previous lesson

What is an FPGA : Field Programmable Gate Array?

… from the previous lesson

What is an FPGA : Field Programmable Gate Array?

FPGA - Wikipedia

https://en.wikipedia.org/wiki/Field-programmable_gate_array

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

… from the previous lesson

What is an FPGA : Field Programmable Gate Array?

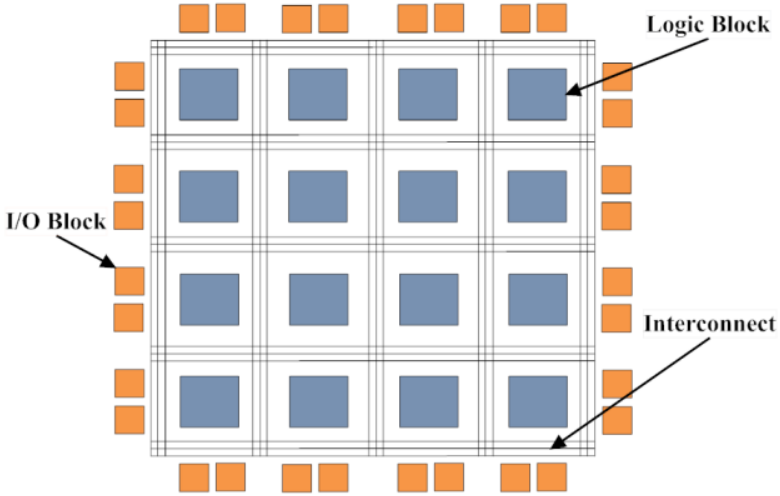
FPGA - Wikipedia

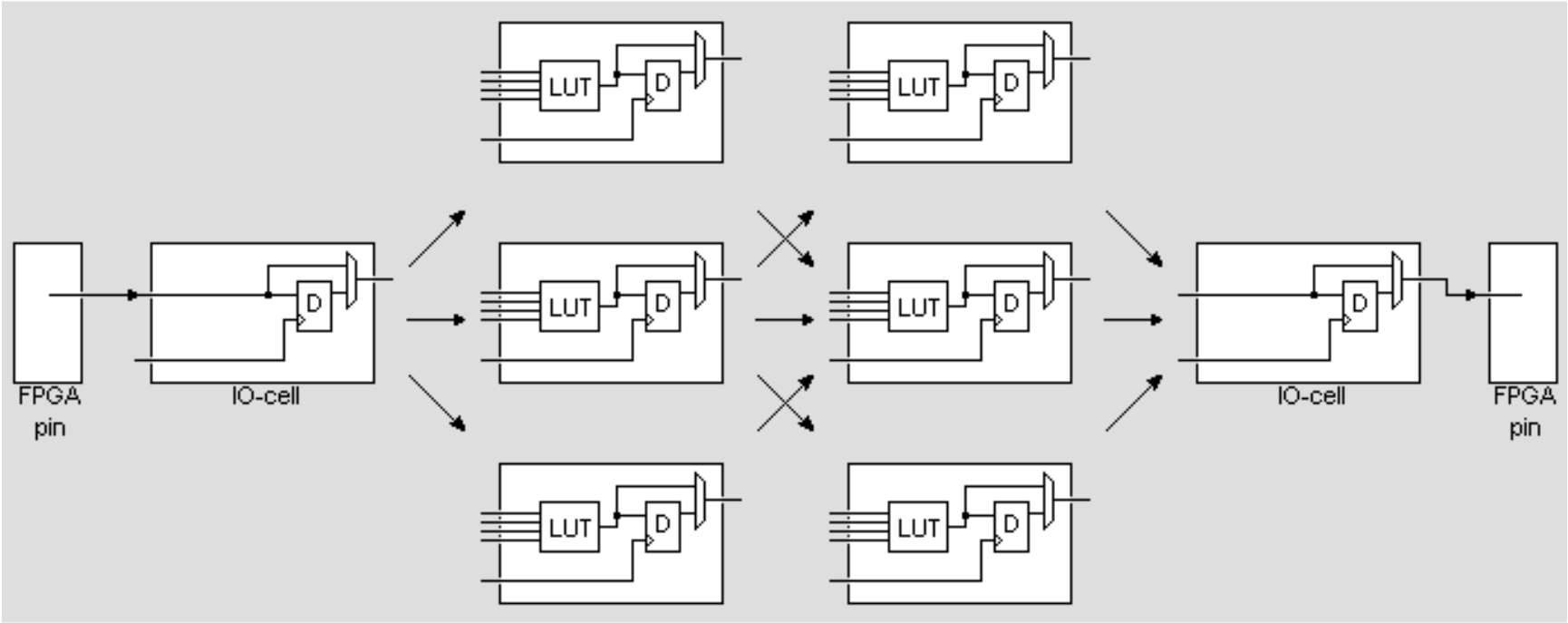
https://en.wikipedia.org/wiki/Field-programmable_gate_array

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

… from the previous lesson

- **FPGA fabric (matrix like structure) made of:**
  - I/O-cells to communicate with outside world
  - Logic cells
    - Look-Up-Table (LUT) to implement combinatorial logic
    - Flip-Flops (D) to implement sequential logic
  - Interconnect network between logic resources
  - Clock tree to distribute the clock signals

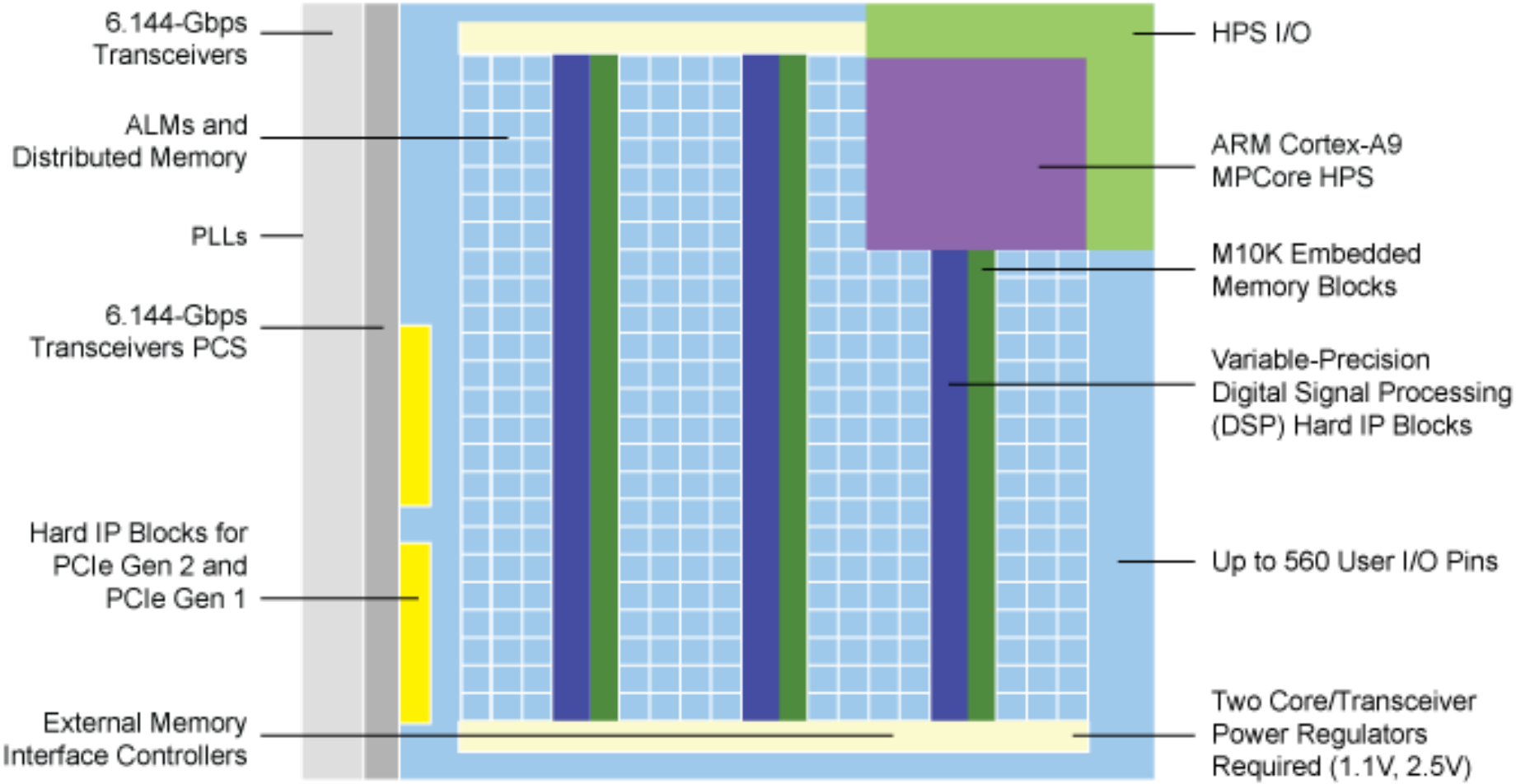




… from the previous lesson

- **But it also features Hard Blocks:**

Example of FPGA architecture



Advanced FPGA design

ISOTDAQ 2016 @ Rehovot (Israel)

29/01/2016

Outline:

- ...from the previous lesson
- **Key concepts about HDL**
- FPGA gateware (firmware) design work flow
- Summary

Manoel Barros Marin

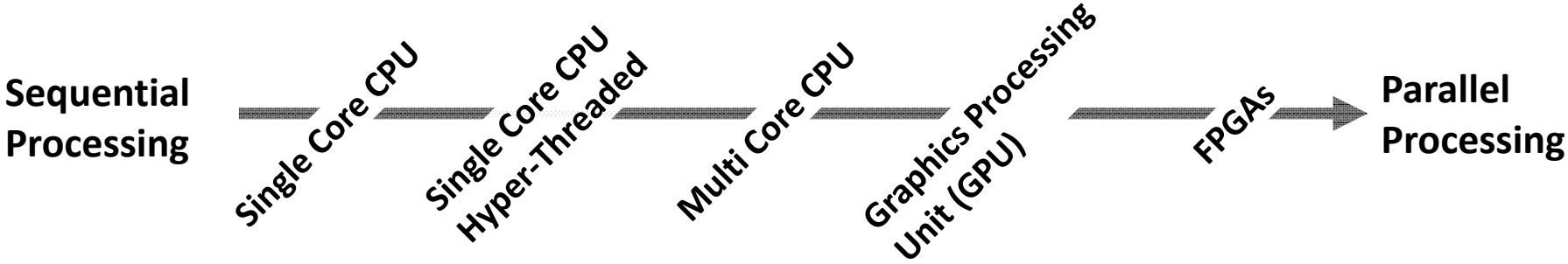
BE-BI-QP

Key concepts about FPGA design

FPGA gateware (firmware) design is NOT programming

Key concepts about FPGA design

FPGA gateware (firmware) design is NOT programming



- **Programming**
  - Code is written and translated into instructions
  - Instructions are executed sequentially by the CPU(s)
  - Parallelism is achieved by running instructions on multiple threads/cores
  - Processing structures and instructions sets are fixed by the architecture of the system

VS.

- **FPGA gateware (firmware) design**
  - No fixed architecture, the system is built according to the task
  - Building is done by describing/defining system elements and their relations
  - Intrinsically parallel, sequential behaviour is achieved by registers and Finite-State-Machines (FSMs)
  - Description done by schematics or a hardware description language (HDL)

Key concepts about HDL

HDL are used for describing HARDWARE

Key concepts about HDL

HDL are used for describing HARDWARE

- Example of a WAIT statement (Programming Language VS. HDL)

Key concepts about HDL

HDL are used for describing HARDWARE

- **Example of a WAIT statement (Programming Language VS. HDL)**
  - In programming language (e.g. C) (Unix, #include <unistd.h>)

```
sleep(5); // sleep 5 seconds
```

  - In HDL (e.g. VHDL):
    - Not synthesizable (only for simulation test benches)

```
wait for 5 sec;  -- handy for TB clocks
```

    - Synthesizable (for simulation and/or FPGA implementation)


```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin  -- process
  if delay_rst = '1' then
    s_count    <= delay_ld_value;
    s_delay_done <= '0';
  elsif rising_edge(delay_clk) then
    if delay_ena = '1' then
      if delay_ld = '1' then
        s_count <= delay_ld_value;
      else
        s_count <= s_count - 1;
      end if;
    end if;
    if s_count = 0 then
      s_delay_done <= '1';
    else
      s_delay_done <= '0';
    end if;
  end if;
 end process;
```

Key concepts about HDL

HDL are used for describing HARDWARE

- **Example of a WAIT statement (Programming Language VS. HDL)**
  - In programming language (e.g. C) (Unix, #include <unistd.h>)

```
sleep(5); // sleep 5 seconds
```

  - In HDL (e.g. VHDL):
    - Not synthesizable (only for simulation test benches)

```
wait for 5 sec;  -- handy for TB clocks
```

    - Synthesizable (for simulation and/or FPGA implementation)


```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin  -- process
  if delay_rst = '1' then
    s_count    <= delay_ld_value;
    s_delay_done <= '0';
  elsif rising_edge(delay_clk) then
    if delay_ena = '1' then
      if delay_ld = '1' then
        s_count <= delay_ld_value;
      else
        s_count <= s_count - 1;
      end if;
    end if;
    if s_count = 0 then
      s_delay_done <= '1';
    else
      s_delay_done <= '0';
    end if;
  end if;
 end process;
```

HDL to RTL

Register Transfer Level (RTL)

http://en.wikipedia.org/wiki/Register-transfer_level

A design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between registers and logical operations performed on those signals

Key concepts about HDL

HDL are used for describing HARDWARE

- **Example of a WAIT statement (Programming Language VS. HDL)**
  - In programming language (e.g. C) (Unix, #include <unistd.h>)

```
sleep(5); // sleep 5 seconds
```

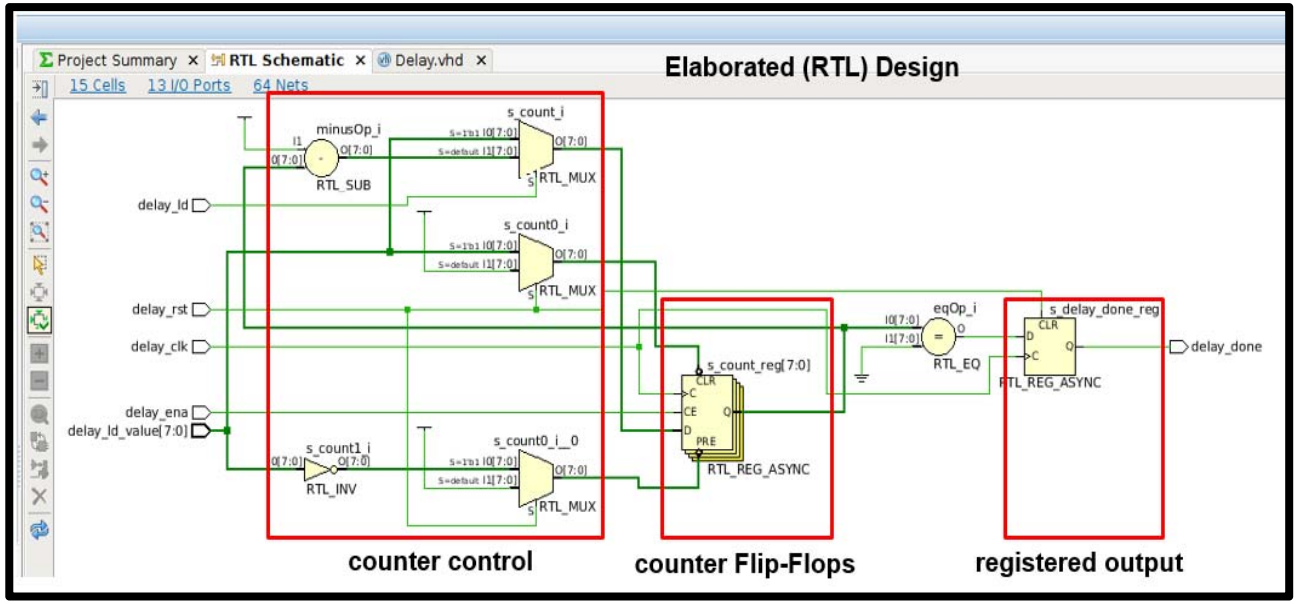
  - In HDL (e.g. VHDL):
    - Not synthesizable (only for simulation test benches)

```
wait for 5 sec;  -- handy for TB clocks
```

    - Synthesizable (for simulation and/or FPGA implementation)


```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin  -- process
  if delay_rst = '1' then
    s_count    <= delay_ld_value;
    s_delay_done <= '0';
  elsif rising_edge(delay_clk) then
    if delay_ena = '1' then
      if delay_ld = '1' then
        s_count <= delay_ld_value;
      else
        s_count <= s_count - 1;
      end if;
    end if;
    if s_count = 0 then
      s_delay_done <= '1';
    else
      s_delay_done <= '0';
    end if;
  end if;
 end process;
```

HDL to RTL



Key concepts about HDL

HDL are used for describing HARDWARE

- **Example of a WAIT statement (Programming Language VS. HDL)**
  - In programming language (e.g. C) (Unix, #include <unistd.h>)

```
sleep(5); // sleep 5 seconds
```

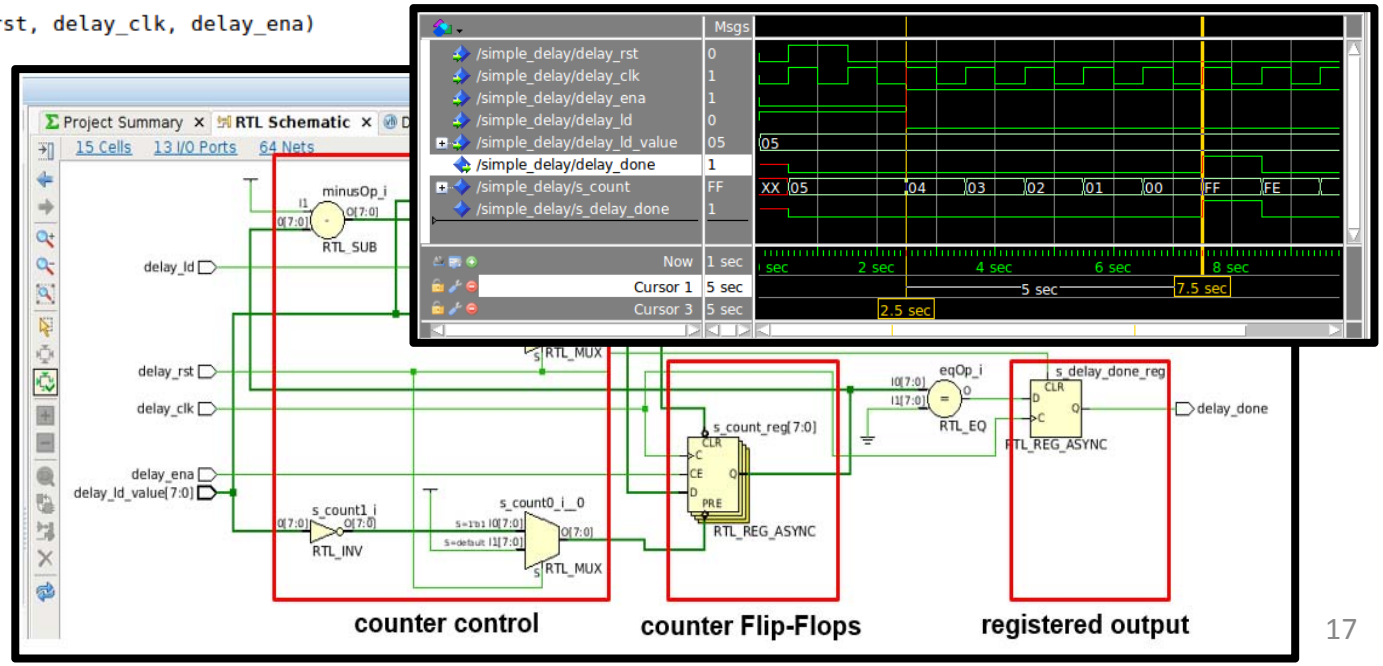
  - In HDL (e.g. VHDL):
    - Not synthesizable (only for simulation test benches)

```
wait for 5 sec;  -- handy for TB clocks
```

    - Synthesizable (for simulation and/or FPGA implementation)


```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin  -- process
  if delay_rst = '1' then
    s_count    <= delay_ld_value;
    s_delay_done <= '0';
  elsif rising_edge(delay_clk) then
    if delay_ena = '1' then
      if delay_ld = '1' then
        s_count <= delay_ld_value;
      else
        s_count <= s_count - 1;
      end if;
    end if;
    if s_count = 0 then
      s_delay_done <= '1';
    else
      s_delay_done <= '0';
    end if;
  end if;
 end process;
```

HDL to RTL



Key concepts about HDL

HDL are used for describing HARDWARE

- **Example of a WAIT statement (Programming Language VS. HDL)**
  - In programming language (e.g. C) (Unix, #include <unistd.h>)

```
sleep(5); // sleep 5 seconds
```

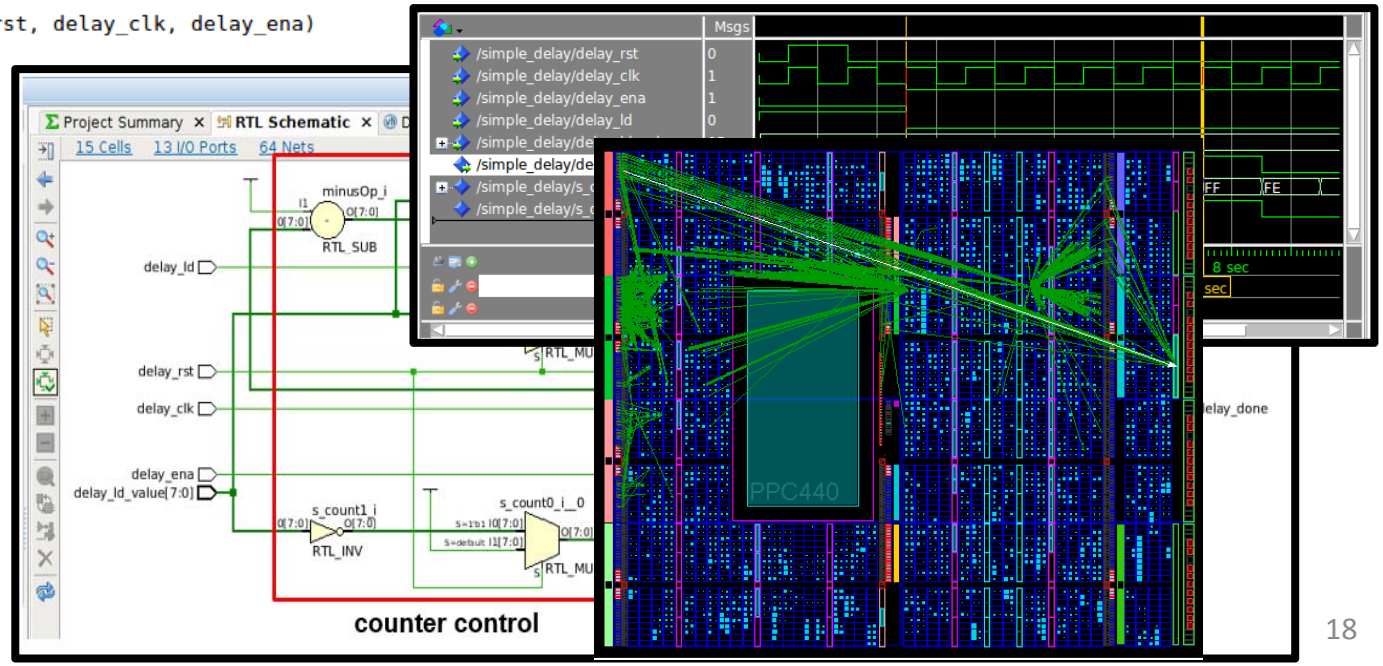
  - In HDL (e.g. VHDL):
    - Not synthesizable (only for simulation test benches)

```
wait for 5 sec;  -- handy for TB clocks
```

    - Synthesizable (for simulation and/or FPGA implementation)


```
simple_delay_counter : process (delay_rst, delay_clk, delay_ena)
begin  -- process
  if delay_rst = '1' then
    s_count    <= delay_ld_value;
    s_delay_done <= '0';
  elsif rising_edge(delay_clk) then
    if delay_ena = '1' then
      if delay_ld = '1' then
        s_count <= delay_ld_value;
      else
        s_count <= s_count - 1;
      end if;
    end if;
    if s_count = 0 then
      s_delay_done <= '1';
    else
      s_delay_done <= '0';
    end if;
  end if;
 end process;
```

HDL to RTL



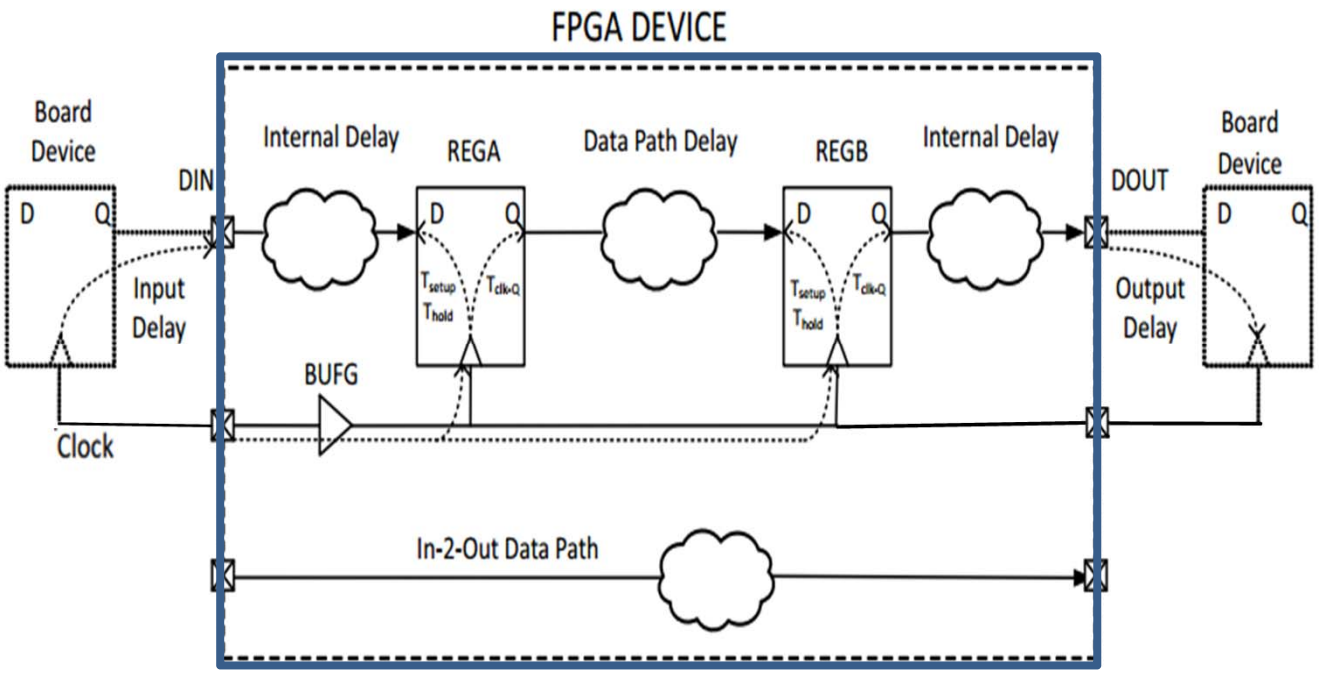
Key concepts about HDL

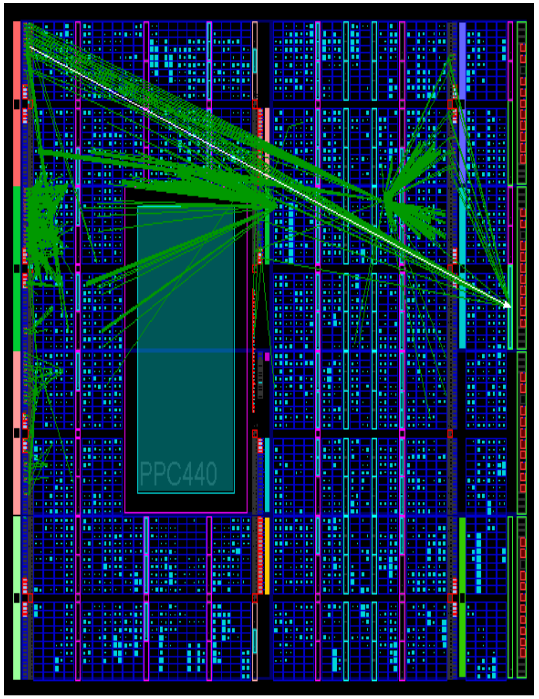
Timing in FPGA gateware (firmware) design is critical

Key concepts about HDL

Timing in FPGA gateware (firmware) design is critical

- Data propagates in the form of electrical signals through the FPGA



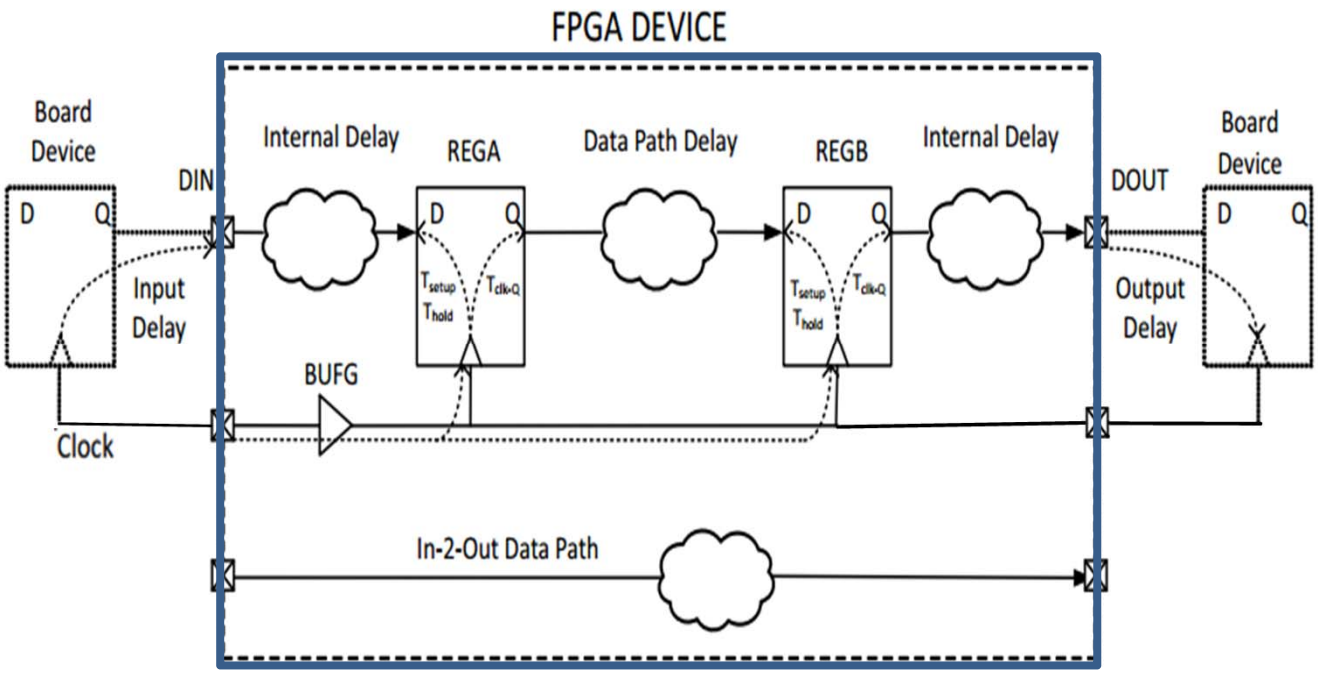


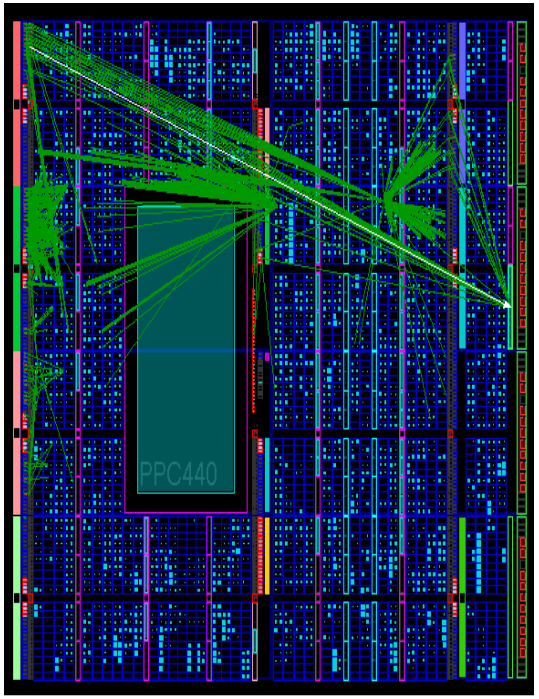
Synthesized RTL (Netlist) is implemented into FPGA

Key concepts about HDL

Timing in FPGA gateware (firmware) design is critical

- Data propagates in the form of electrical signals through the FPGA





Synthesized RTL (Netlist) is implemented into FPGA

- If these signals do not arrive to their destination on time…

The consequences may be catastrophic!!!

Key concepts about HDL

When designing FPGA gateware (firmware) you have to think HARD…

Key concepts about HDL

When designing FPGA gateware (firmware) you have to think HARDWARE

Advanced FPGA design

ISOTDAQ 2016 @ Rehovot (Israel)

29/01/2016

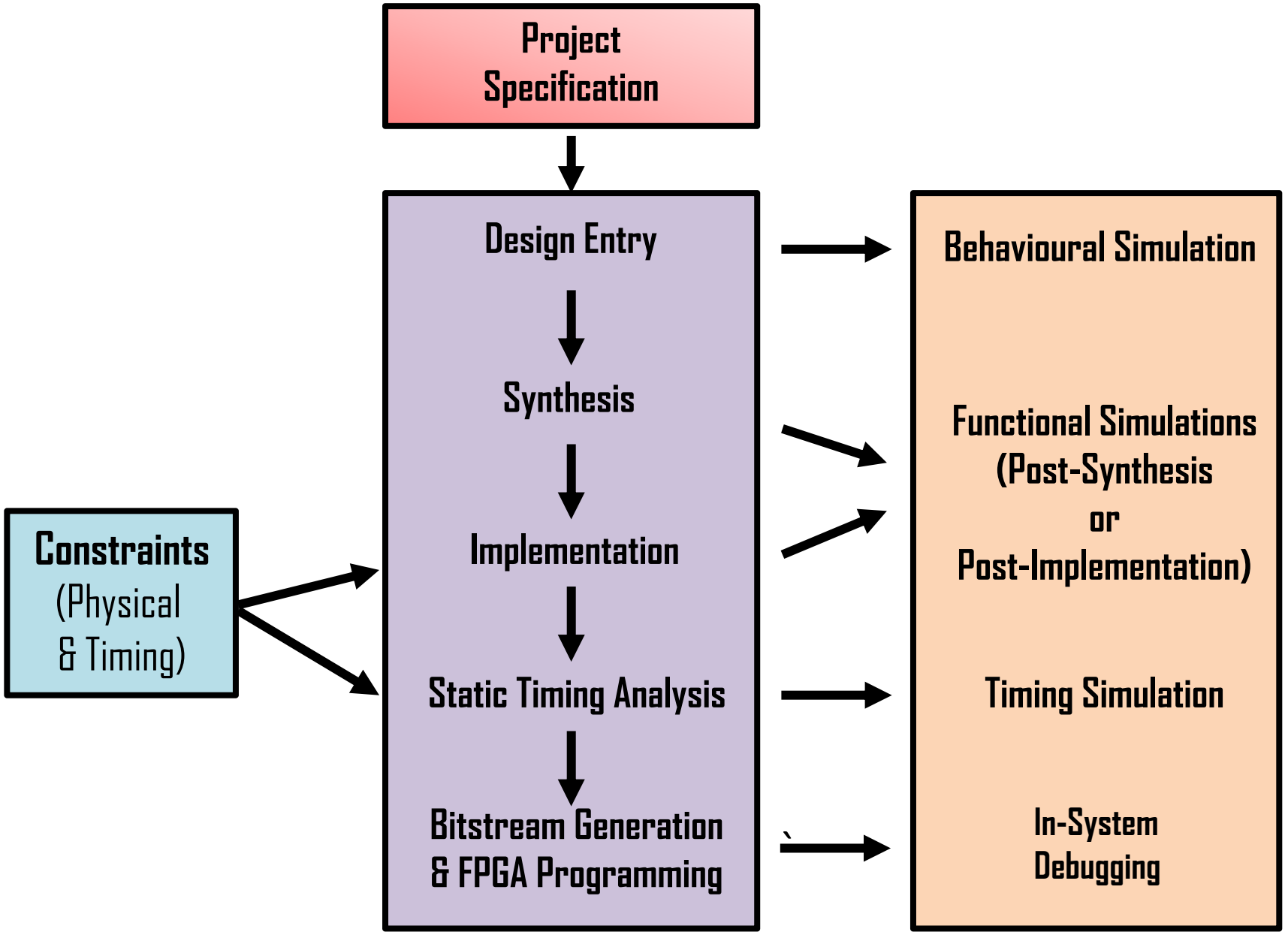
Outline:

- ...from the previous lesson
- Key concepts about HDL
- **FPGA gateware (firmware) design work flow**
- Summary

Manoel Barros Marin

BE-BI-QP

FPGA gateware (firmware) design work flow



FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

The rest of the design process is based on it!!!

FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- Gather requirements from the users

The rest of the design process is based on it!!!

FPGA gateware (firmware) design work flow

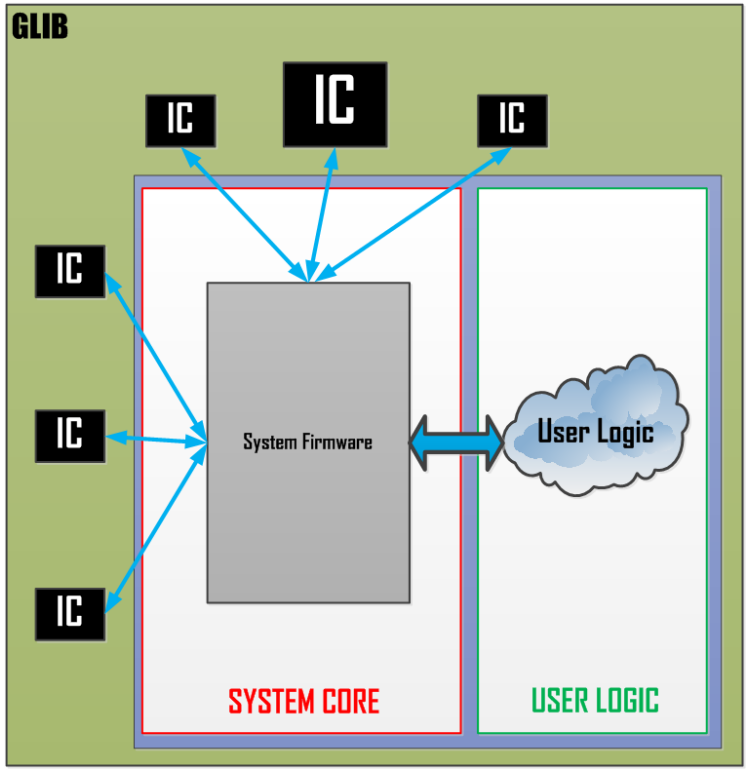
Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)

The rest of the design process is based on it!!!

Example of General Purpose Gateware



FPGA gateware (firmware) design work flow

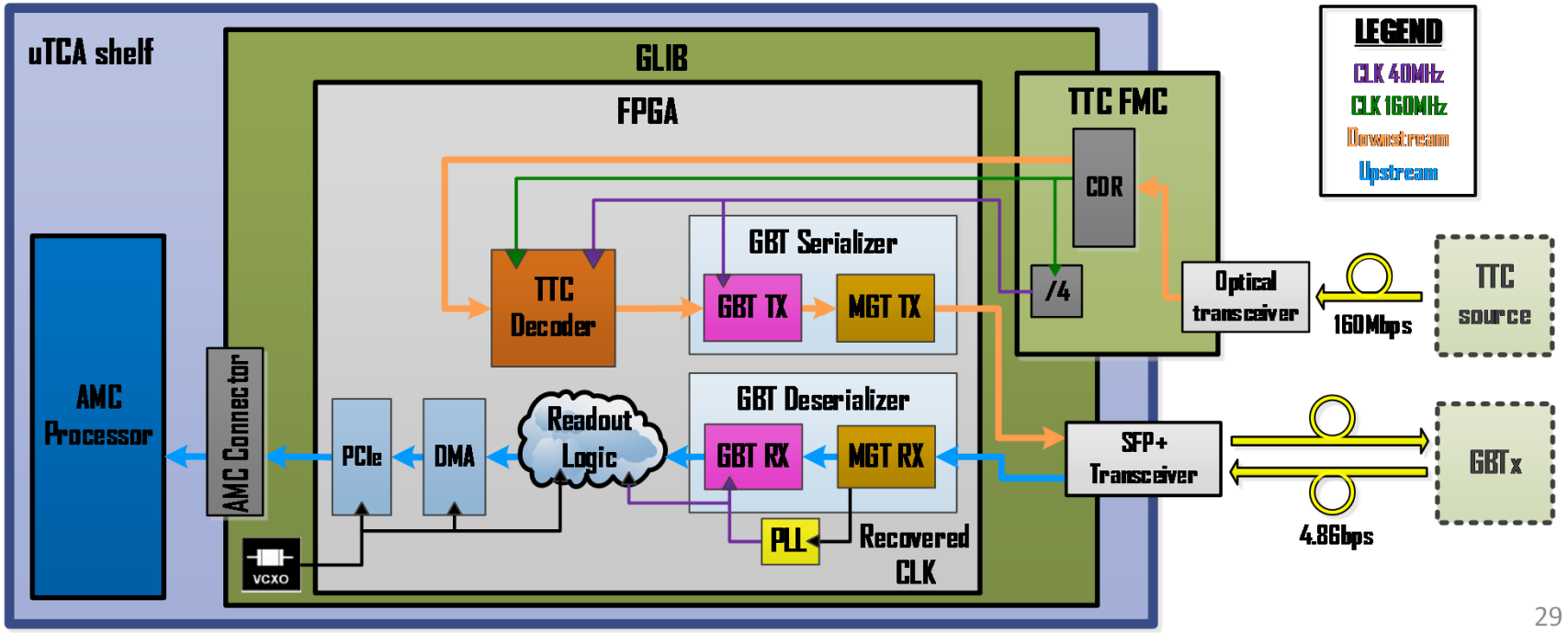
Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)

The rest of the design process is based on it!!!

Example of Application Specific Gateware



FPGA gateware (firmware) design work flow

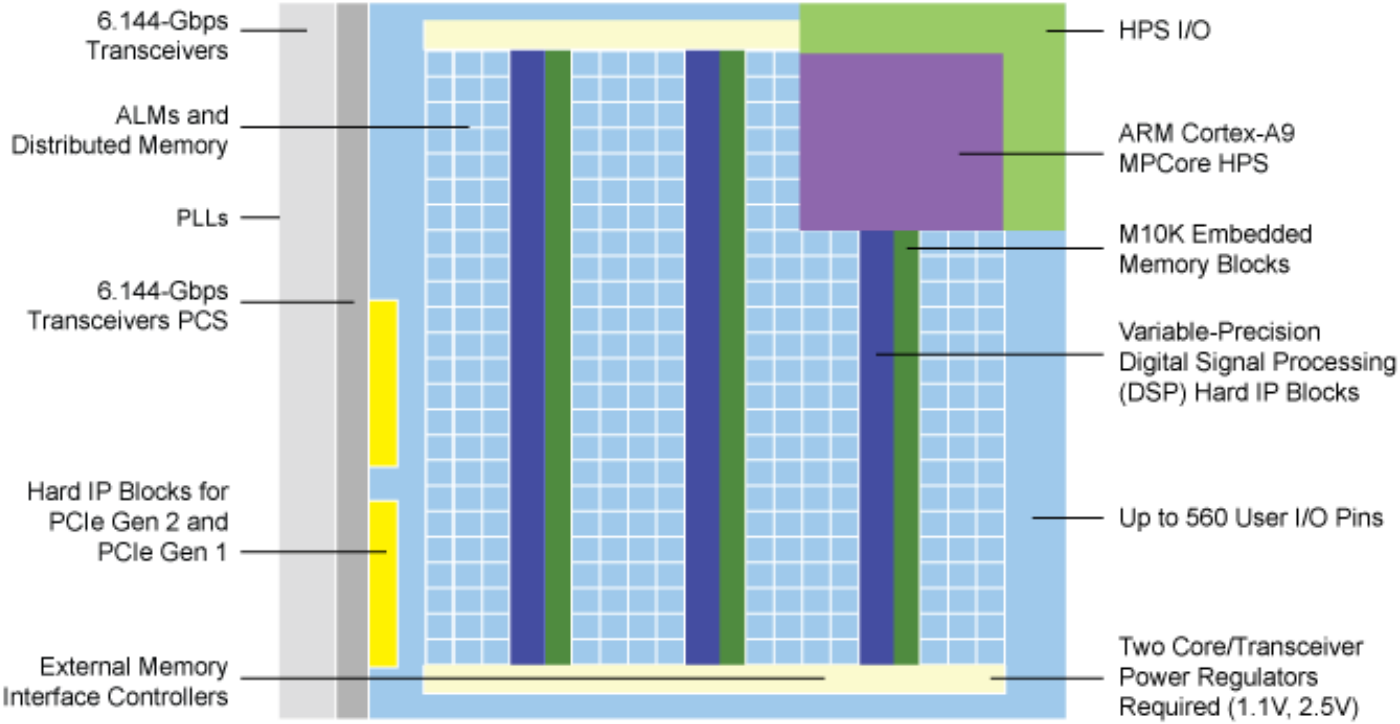
Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)

The rest of the design process is based on it!!!

Example of FPGA Architecture



FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)

The rest of the design process is based on it!!!

XILINX ALL PROGRAMMABLE™

ALTERA now part of Intel

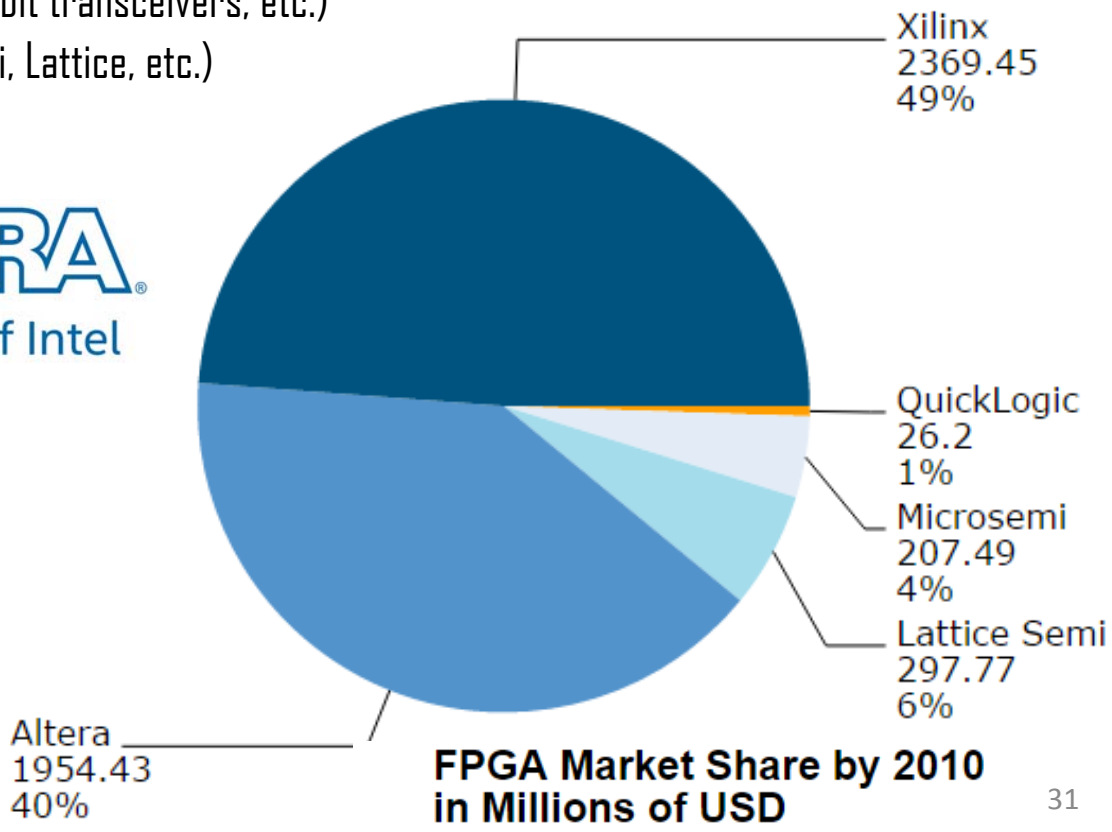
ATMEL

Microsemi

LATTICE SEMICONDUCTOR

QuickLogic

Small FPGA vendors may target specific markets
(e.g. Microsemi offers high reliable FPGAs, etc..)



FPGA Market Share by 2010 in Millions of USD

FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))

The rest of the design process is based on it!!!

Example of Custom Board

Example of COST board (Xilinx Devkit)

(*) Commercial off-the-shelf (COTS)

FPGA gateware (firmware) design work flow

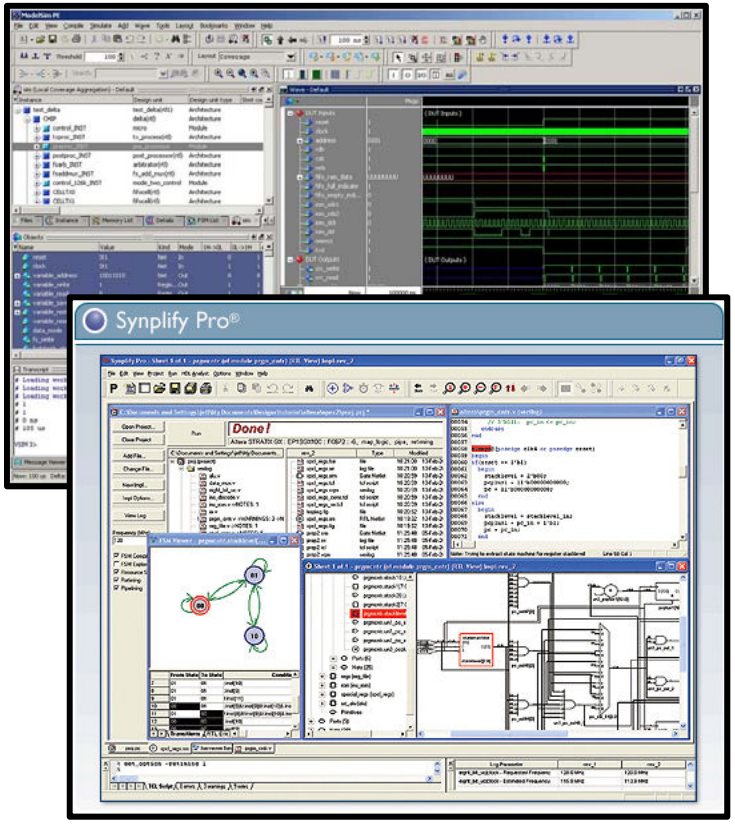
Project Specification

This is the most critical step…

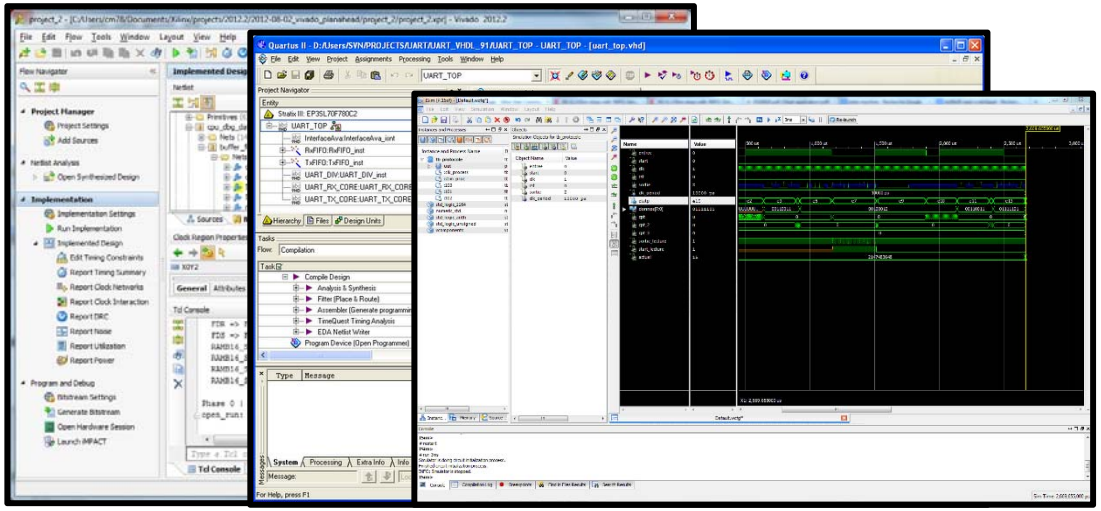
- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)

The rest of the design process is based on it!!!

Example of Commercial Tools



Example of FPGA Vendor Tools



FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users** The rest of the design process is based on it!!!
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)

FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users** The rest of the design process is based on it!!!
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)

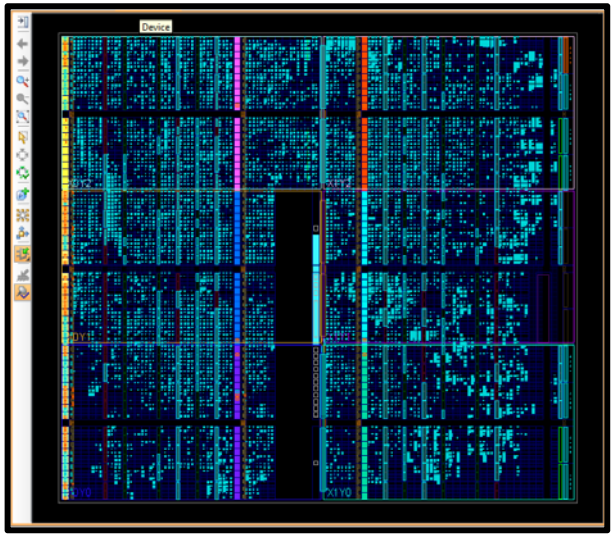
FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)

The rest of the design process is based on it!!!



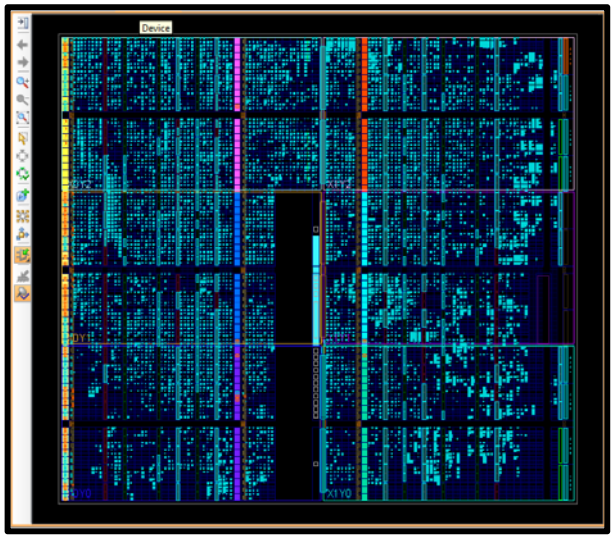
FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)

The rest of the design process is based on it!!!



FPGA gateware (firmware) design work flow

Project Specification

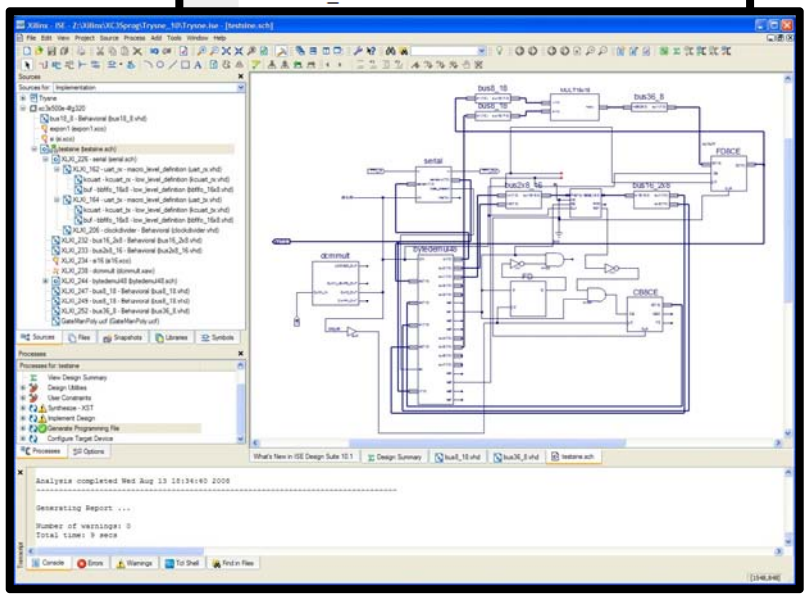
This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)
  - Design language (Schematics or HDL (e.g. VHDL, etc.))

The rest of the design process is based on it!!!

Examples of Design Languages

```
31
32  entity XuLA_2 is
33      Port (  PB1          : in  STD_LOGIC;
34              PB2          : in  STD_LOGIC;
35              PB3          : in  STD_LOGIC;
36              PB4          : in  STD_LOGIC;
37              LED1         : out  STD_LOGIC;
38              LED2         : out  STD_LOGIC;
39              LED3         : out  STD_LOGIC;
40              LED4         : out  STD_LOGIC);
41  end XuLA_2;
```



HDL are the most popular for RTL design but…

Schematics may be better in some cases (e.g. Embedded Processor configuration, etc..)

FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users**

The rest of the design process is based on it!!!

- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)
  - Design language (Schematics or HDL (e.g. VHDL, etc.))
  - Coding convention

Your code should be readable

Example of Coding Convention

description	extension	example
variable	prefix v	v_Buffer
alias	prefix a	a_Bit5
constant	prefix c	c_Lenght
type definition	prefix t	t_MyType
generics	prefix g	g_Width

FPGA gateware (firmware) design work flow

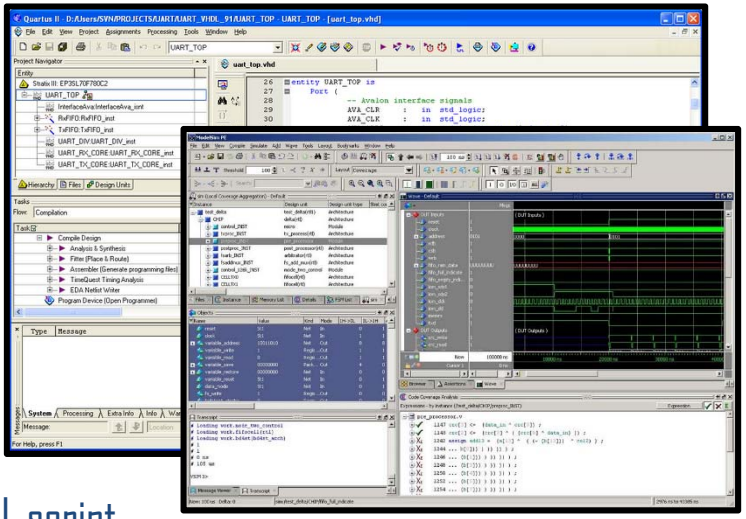
Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)
  - Design language (Schematics or HDL (e.g. VHDL, etc.))
  - Coding convention
  - Software interface (GUI, Scripts or both)

The rest of the design process is based on it!!!

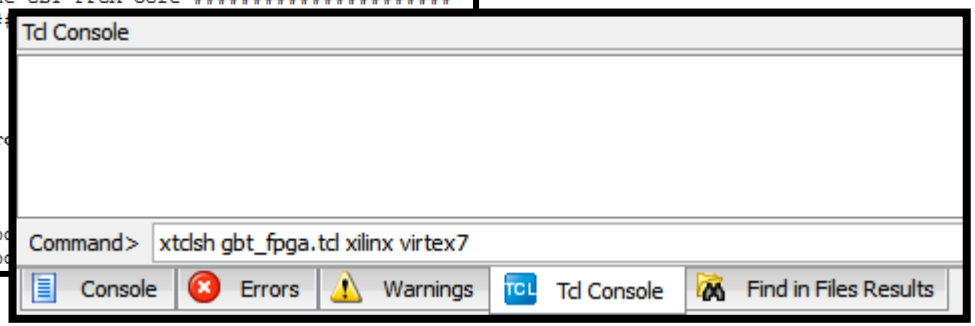
Example of GUIs



Example of TCL script

```
45  ##################################################################################################
46  ##################### Commands for Adding the Source Files of the GBT-FPGA Core ##################
47  ##################################################################################################
48
49  ## Comment: Adding Common files:
50
51  puts "->"
52  puts "-> Adding common files of the GBT-FPGA Core to the ISE pr
53  puts "->"
54
55  xfile add $SOURCE_PATH/gbt_bank/core_sources/gbt_rx/gbt_rx.vhd
56  xfile add $SOURCE_PATH/gbt_bank/core_sources/gbt_rx/gbt_rx_deco
57  xfile add $SOURCE_PATH/gbt_bank/core_sources/gbt_rx/gbt_rx_deco
```

Xilinx ISE TCL console



FPGA gateware (firmware) design work flow

Project Specification

This is the most critical step…

- **Gather requirements from the users**
- **Specify:**
  - Target application (Specific or General purpose)
  - Main features (e.g. System bus, Multi-gigabit transceivers, etc.)
  - FPGA vendor (e.g. Xilinx, Altera, Microsemi, Lattice, etc.)
  - Electronic board (Custom or COST (*))
  - Development tools (FPGA vendor or Commercial)
  - Optimization (Speed, Area, Power or default)
  - Design language (Schematics or HDL (e.g. VHDL, etc.))
  - Coding convention
  - Software interface (GUI, Scripts or both)
  - Use of files repository (SVN, GIT, etc.. or none)

The rest of the design process is based on it!!!

FPGA gateware (firmware) design work flow

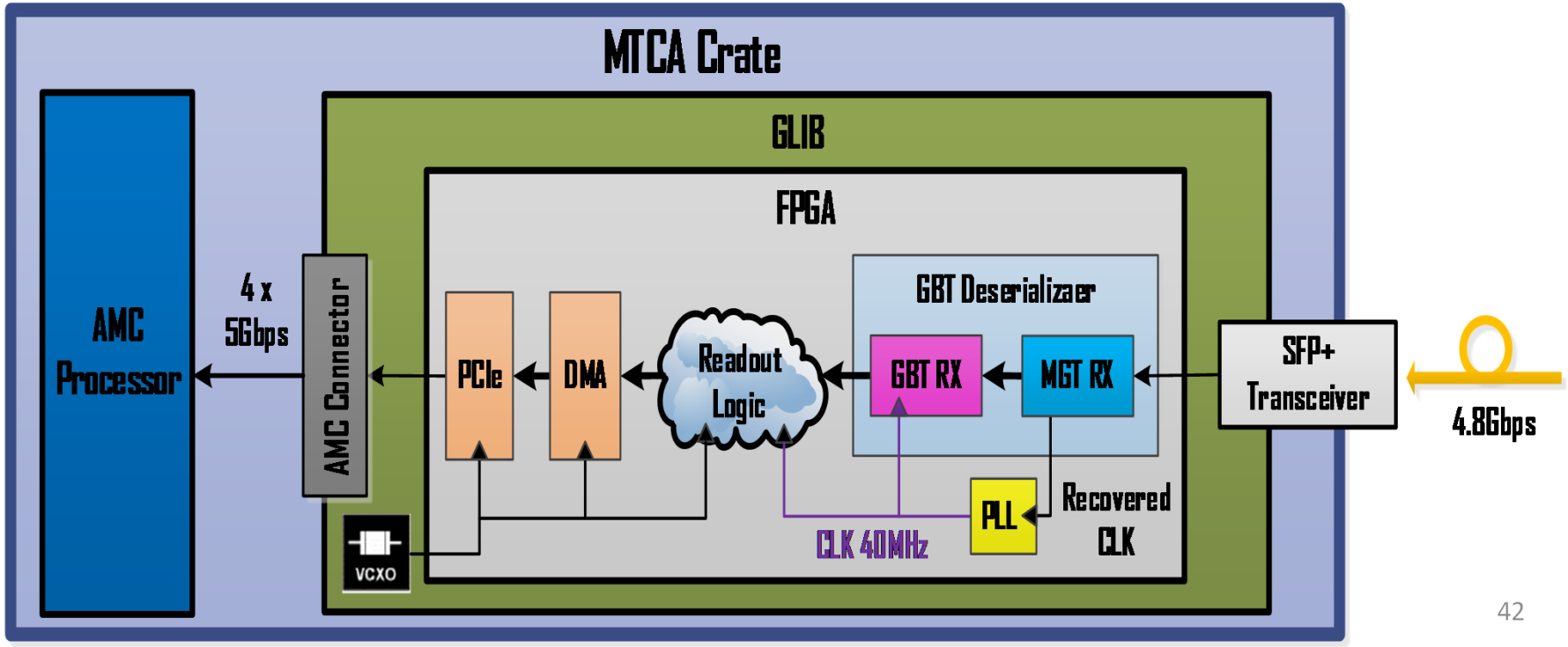
Project Specification

This is the most critical step...

- **Block diagram of the system**
  - Include the FPGA logic...
  - ... but also the on-board devices and related devices
  - May combine different abstraction levels

The rest of the design process is based on it!!!

Example of system block diagram



FPGA gateware (firmware) design work flow

Project Specification

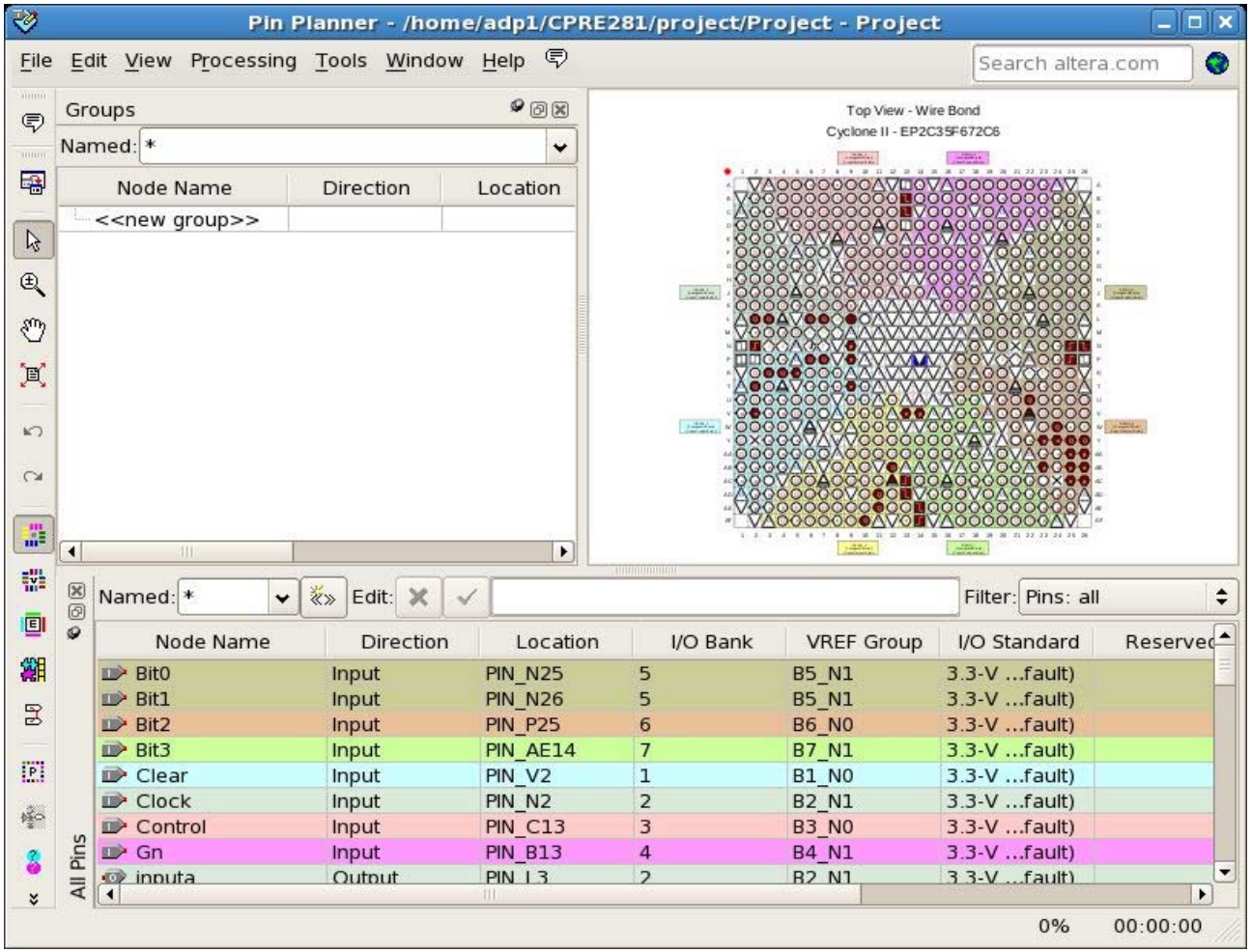
This is the most critical step…

- **Pin planning**

The rest of the design process is based on it!!!

Pin assignments are one type of Location Constraints

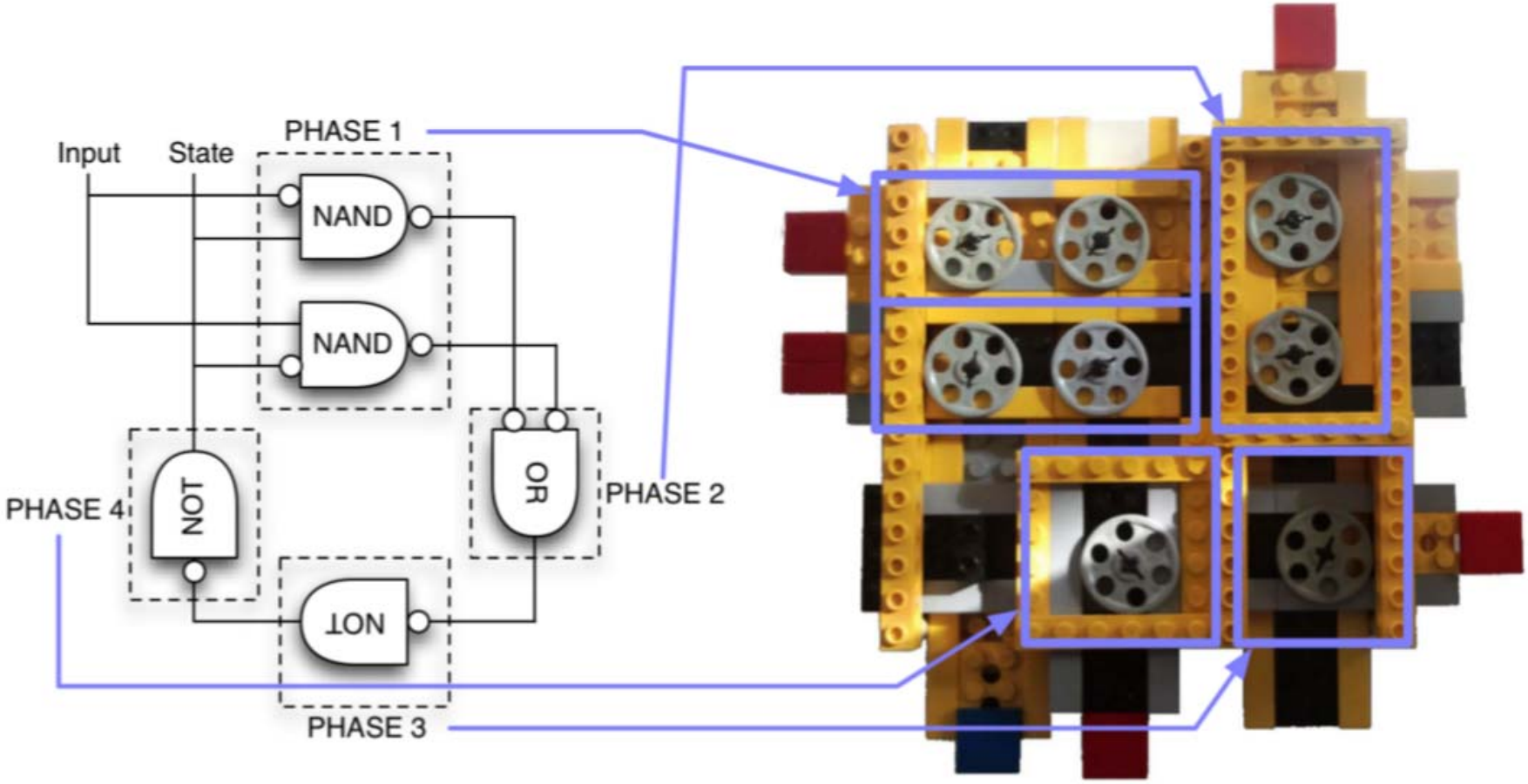
Critical for Custom Boards!!!



Example of Pin Planner GUI

FPGA gateware (firmware) design work flow

Design Entry

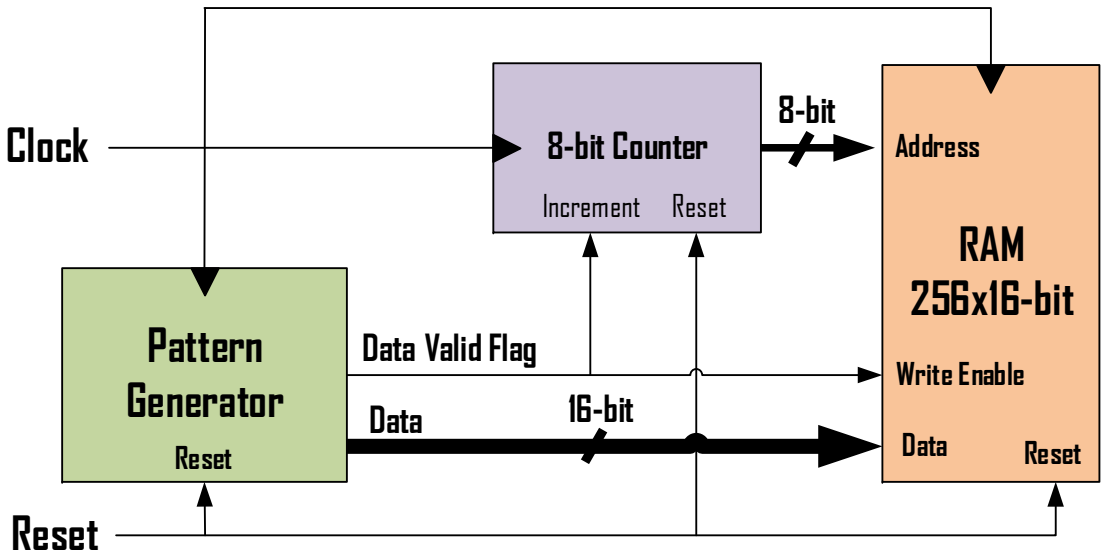


FPGA gateware (firmware) design work flow

Design Entry: Modularity & Reusability

- **Your system should be Modular**
  - Design at RTL level (think hard…ware)
  - Well defined clocks and resets schemes
  - Separated Data & Control paths
  - Multiple instantiations

Good example of Modular System



- **Your code should be Reusable**
  - Add primitives (and modules) to the system by inference when possible
  - Use parameters in your code (e.g. generics in VHDL, parameters in Verilog, etc.)
  - Centralise parameters in external files (e.g. packages in VHDL, headers in Verilog, etc.)
  - Use configurable modules interfaces when possible (e.g. parametrised vectors, records in VHDL, etc.)
  - Use standard features (e.g. I2C, Wishbone, etc.)
  - Use standard IP Cores (e.g. from www.OpenCores.org, etc.)
  - Avoid vendor specific IP Cores when possible
  - Talk with your colleagues and see what other FPGA designers are doing

FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

Synthesizable code is intended for FPGA implementation

- **Use non-synthesizable HLD statements only in simulation test benches**

  A fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis.

  From book "Advanced FPGA Design" by Steve Kilts (Copyright © 2007 John Wiley & Sons, Inc.)

FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

Synthesizable code is intended for FPGA implementation

- **Use non-synthesizable HLD statements only in simulation test benches**

  A fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis.

  From book "Advanced FPGA Design" by Steve Kilts (Copyright © 2007 John Wiley & Sons, Inc.)

- **The RTL synthesis tool is expecting a synchronous design…**

FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

Synthesizable code is intended for FPGA implementation

- **Use non-synthesizable HLD statements only in simulation test benches**

  A fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis.

  From book "Advanced FPGA Design" by Steve Kilts (Copyright © 2007 John Wiley & Sons, Inc.)

- **The RTL synthesis tool is expecting a synchronous design…**

But what is a synchronous design???

FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

Synthesizable code is intended for FPGA implementation

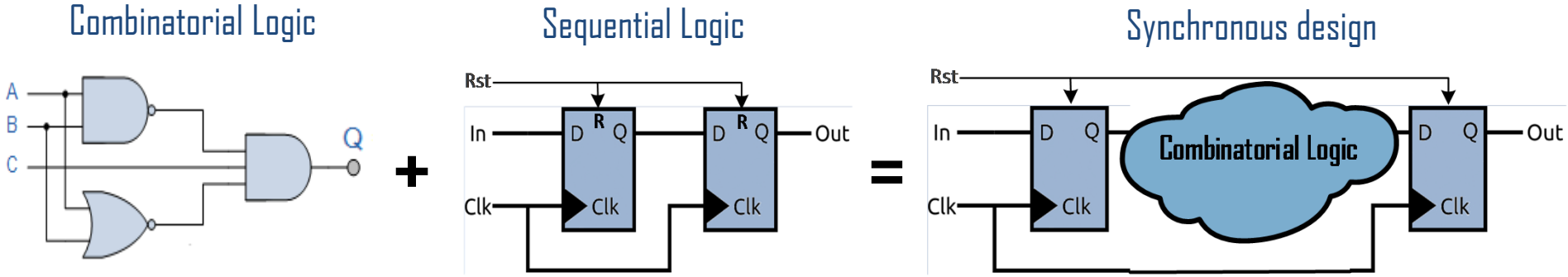
- **Use non-synthesizable HLD statements only in simulation test benches**

  A fundamental guiding principle when coding for synthesis is to minimize, if not eliminate, all structures and directives that could potentially create a mismatch between simulation and synthesis.

  From book "Advanced FPGA Design" by Steve Kilts (Copyright © 2007 John Wiley & Sons, Inc.)

- **The RTL synthesis tool is expecting a synchronous design…**

  Synchronous design is the one compose by combinatorial logic (e.g. logic gates, multiplexors, etc..) and sequential logic (registers that are triggered on the edge of a single clock),



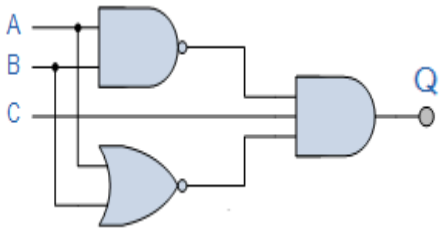
FPGA design tools only analyse synchronous designs!!!

FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

- **Combinatorial logic coding rules**
  - Sensitivity list must include ALL input signals
    Not respecting this may lead to non responsive outputs under changes of input signals
  - ALL output signals must be assigned under ALL possible input conditions
    Not respecting this may lead to undesired latches (asynchronous storage element)
  - No feedback from output to input signals
    Not respecting this may lead to unknown output states (metastability) & undesired latches

Good combinatorial coding for synthesis



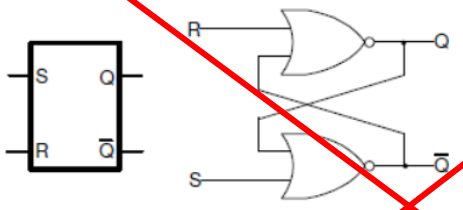
Typical Truth Table

C	B	A	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

```
process(Input_A,Input_B,Input_C)
begin
    Output_nand <= Input_A nand Input_B;
    Output_nor  <= Input_A nor  Input_B;
    --
    Output_Q    <= Output_nand and Input_C and Output_nor;
end process;
```

Bad combinatorial coding for synthesis

Asynchronous Latch



S	R	Q	Q̄
0	0	Latch	
0	1	0	1
1	0	1	0
1	1	Metastable	

```
process(Input_R)
begin
    Output_Q   <= Input_R nor Output_Q_n;
    Output_Q_n <= Input_S nor Output_Q;
end process;
```

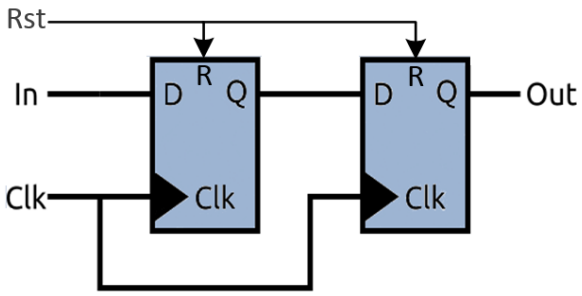
FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

- **Sequential logic coding rules**
  - Only clock signal (and asynchronous set/reset signals when used) in sensitivity list
    Not respecting this may produce undesired combinatorial logic
  - All registers of the sequence must be triggered by the same clock edge (either Rising or Falling)
    Not respecting this may lead to metastability at the output of the registers
  - Include all registers of the sequence in the same reset branch
    Not respecting this may lead to undesired register values after reset

Good sequential coding for synthesis

```
process(Clk,Rst)
begin
    if (Rst = '1') then
        Output_Out <= '0';
        Output_Q   <= '0';
    elsif rising_edge(Clk) then
        Output_Out <= Output_Q;
        Output_Q   <= Input_In;
    end if;
end process;
```



Bad sequential coding for synthesis

```
process(Clk,Rst,Input_In)
begin
    if (Rst = '1') then
        Output_Out <= '0';
    elsif rising_edge(Clk) then
        Output_Out <= Output_Q;
        Output_Q   <= Input_In;
    end if;
end process;
```

FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

- **Synchronous design coding rules:**
  - FULLY synchronous design
    - No combinatorial feedback
    - No asynchronous latches

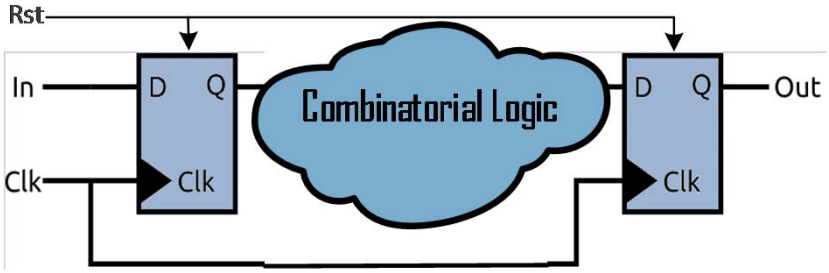
    Not respecting this may lead to incorrect analysis from the FPGA design tool
  - Register ALL output signal

    Not respecting this may lead to uncontrolled length of combinatorial paths
  - Properly design of reset scheme (mentioned later)

    Not respecting this may lead to undesired register values after reset
  - Properly design of clocking scheme (mentioned later)

    Not respecting this may lead to metastability at the output of the registers & Misuse of resources
  - Properly handle Clock Domain Crossings (CDC) (mentioned later)

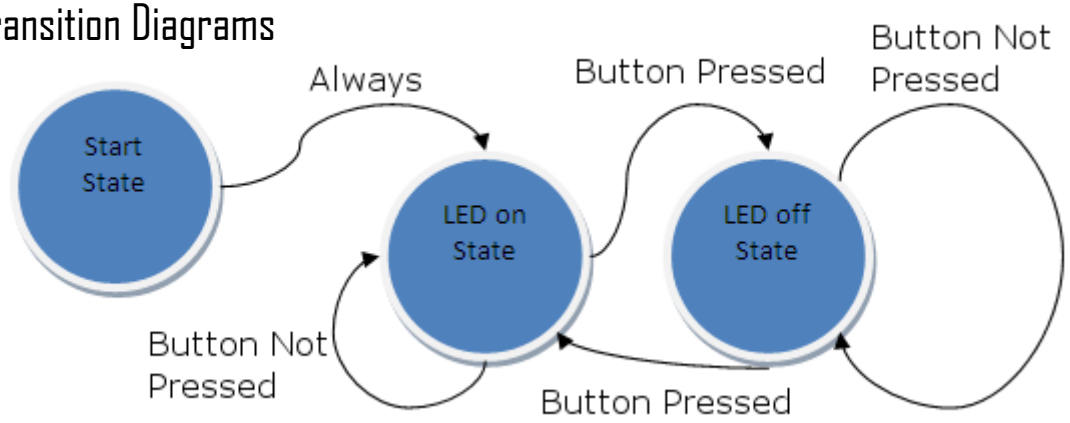
    Not respecting this may lead to metastability at the output of the registers



FPGA gateware (firmware) design work flow

Design Entry: Coding for Synthesis

- **Finite State Machines (FSMs):**
  - Digital logic circuit with a finite number of internal states
  - Widely used for system control
  - Two variants of FSM
    - Moore: Outputs depends only on the current state of the FSM
    - Mealy: Outputs depends only on the current state of the FSM as well as the current values of the inputs
  - Modelled by State Transition Diagrams



  - Many different FSM coding styles (But not all of the are good!!)
  - FSM coding considerations:
    - Synchronize inputs & outputs
    - Outputs may be assigned during states or state transitions
    - Be careful with unreachable/illegal states
    - You can add counters to FSMs

FPGA gateware (firmware) design work flow

Design Entry: Reset Scheme

A bad reset scheme may get you crazy!!!

- **Used to initialize the output of the registers to a know state**
- **It has a direct impact on:**
  - Performance
  - Logic utilization
  - Reliability
- **Different opinions due to…**
- **Different approaches:**
  - Asynchronous

    Pros: No free running clock required, easier timing closure

    Cons: skew, glitches, simulation mismatch, difficult to debug, extra constraints, etc.
  - Synchronous

    Pros: No Skew, No Glitches, No simulation mismatch, Easier to debug, No extra constraints, etc..

    Cons: Free-running clock required, More difficult timing closure
  - No Reset Scheme

    Pros; Easier Routing, Less resources, Easiest timing closure

    Cons: Only reset at power up (in some devices not even that…) **<- In fact, reset is not always needed**
  - Hybrid: Usually in big designs (**Avoid when possible!!!**)

FPGA gateware (firmware) design work flow

Design Entry: Reset Scheme

A bad reset scheme may get you crazy!!!

- **Used to initialize the output of the registers to a know state**
- **It has a direct impact on:**
  - Performance
  - Logic utilization
  - Reliability
- **Different opinions due to…**
- **Different approaches:**
  - Asynchronous

    Pros: No free running clock required, easier timing closure

    Cons: skew, glitches, simulation mismatch, difficult to debug, extra constraints, etc.
  - Synchronous

    Pros: No Skew, No Glitches, No simulation mismatch, Easier to debug, No extra constraints, etc..

    Cons: Free-running clock required, More difficult timing closure
  - No Reset Scheme

    Pros; Easier Routing, Less resources, Easiest timing closure

    Cons: Only reset at power up (in some devices not even that…) **<- In fact, reset is not always needed**
  - Hybrid: Usually in big designs (**Avoid when possible!!!**)

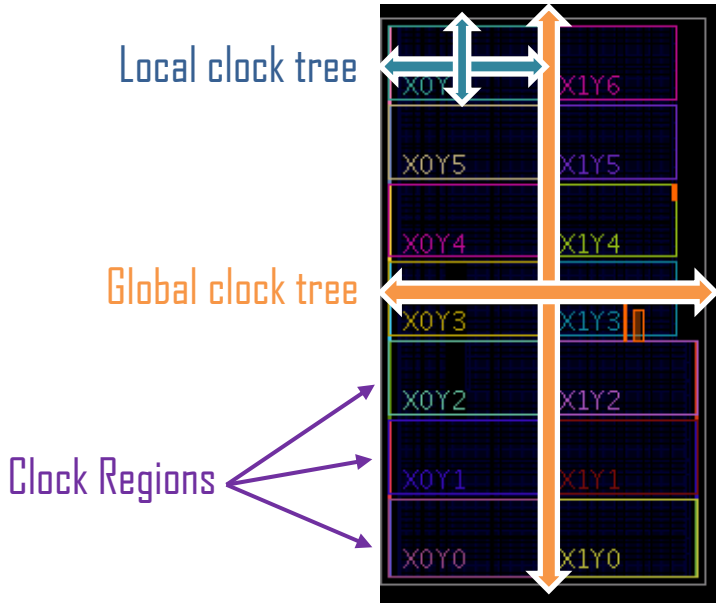
My advise is…
You should use
SYNCHRONOUS RESET
by default

FPGA gateware (firmware) design work flow

Design Entry: Clocks Scheme

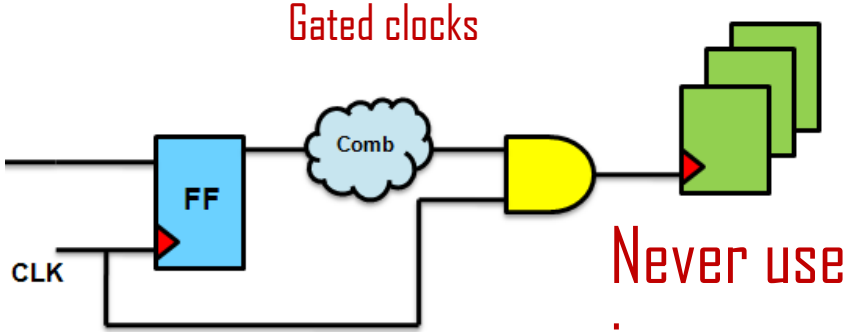
Clocking resources are very precious!!!

- **Clock regions**
- **Clock trees (Global & Local)**
- **Other FPGA clocking resources**
  - Clock capable pins
  - Clock buffers
  - Clock Multiplexors
  - PLLs & DCM



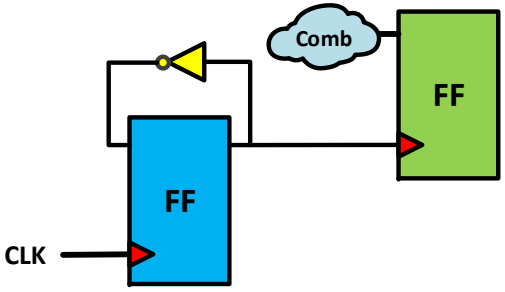
- **Bad practices when designing your clocking scheme**

Gated clocks



Never use these clocks in your system!!!

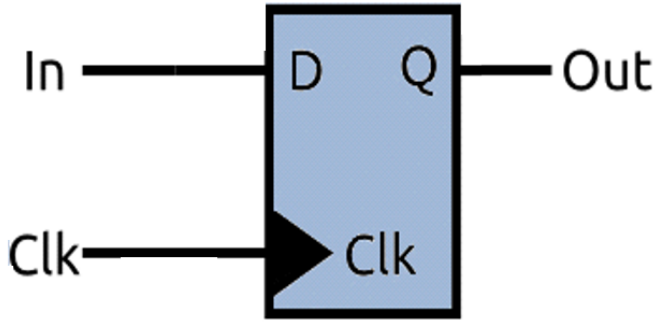
Derived clocks

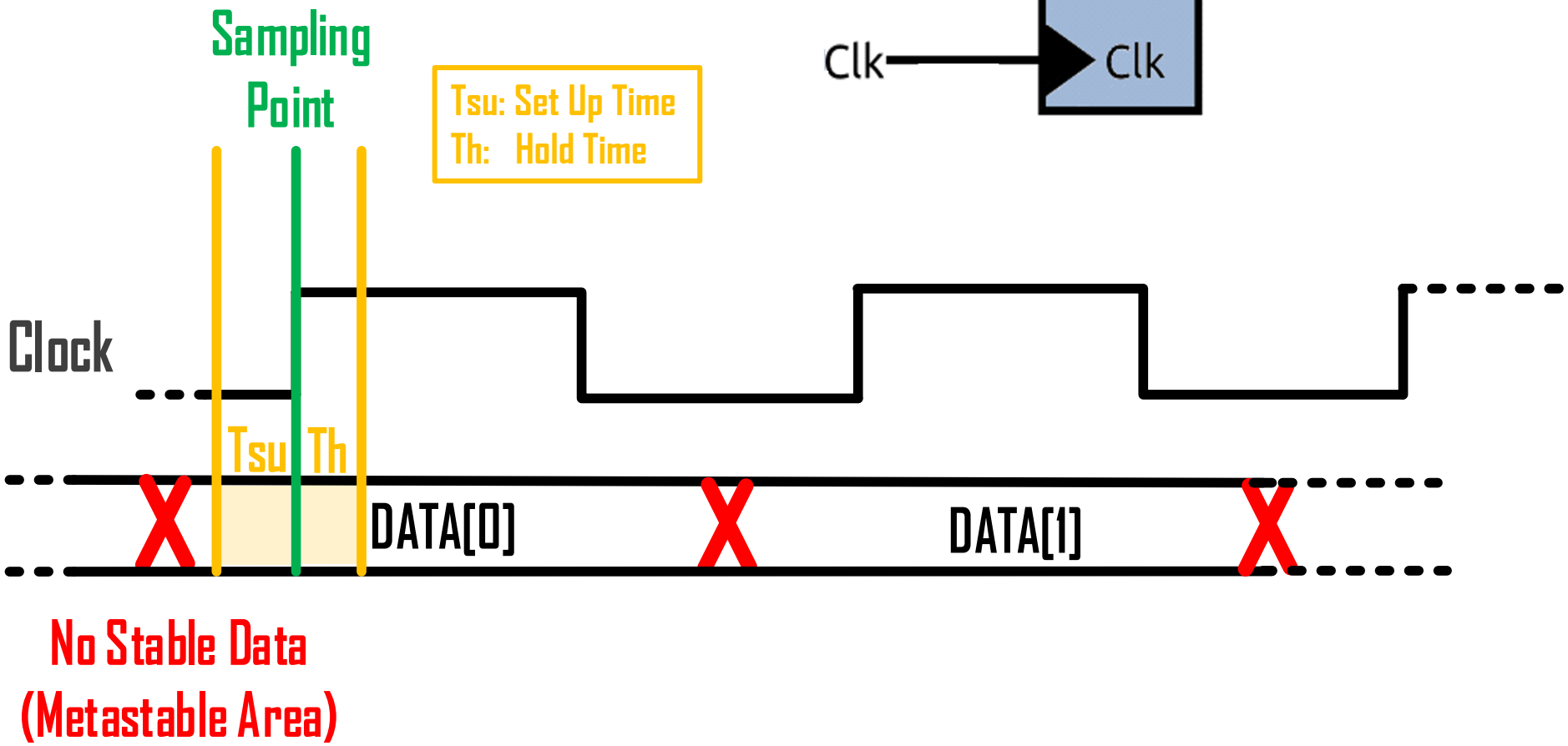


FPGA gateware (firmware) design work flow

Design Entry: Timing

- Sampling

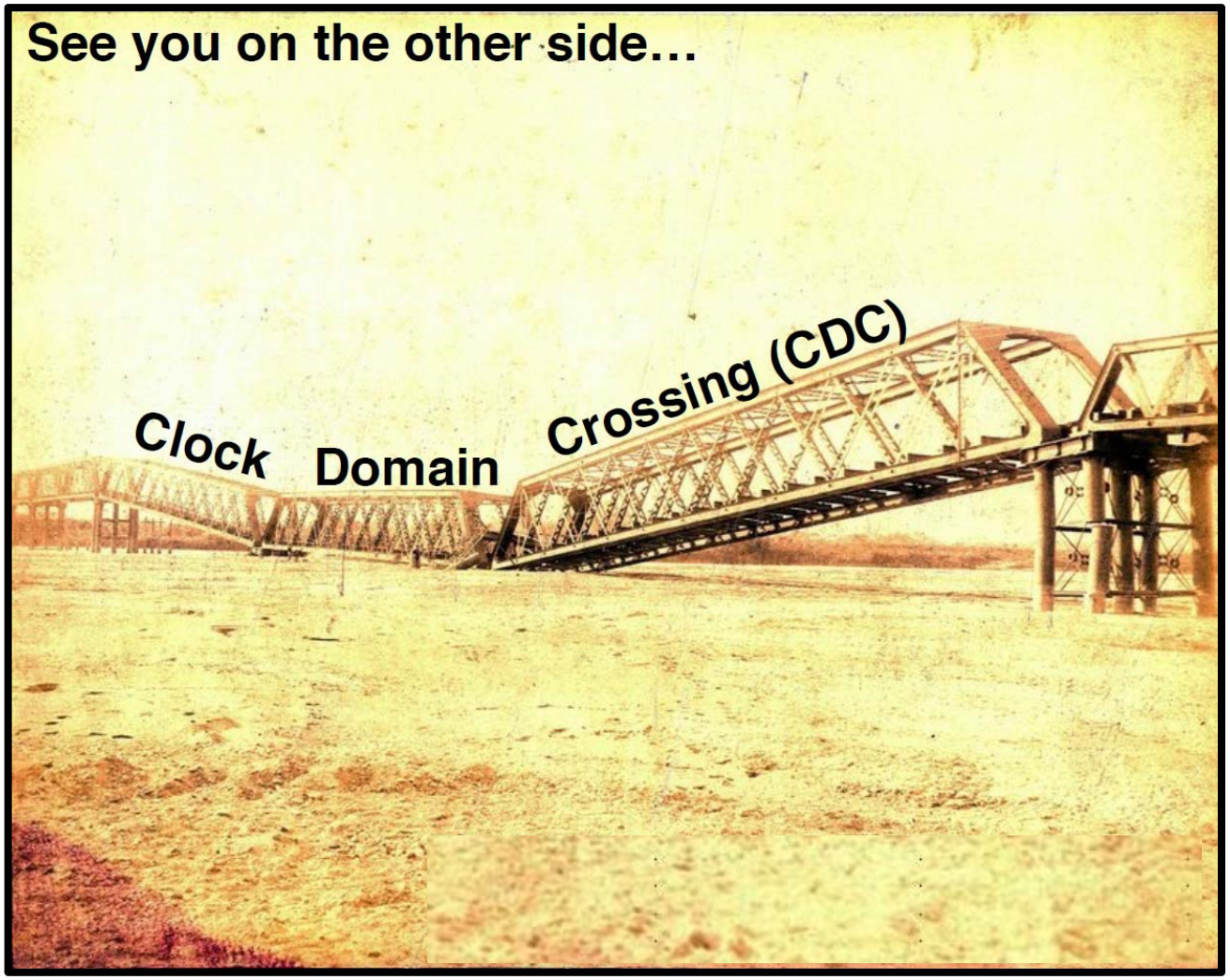




FPGA gateware (firmware) design work flow

Design Entry: Timing

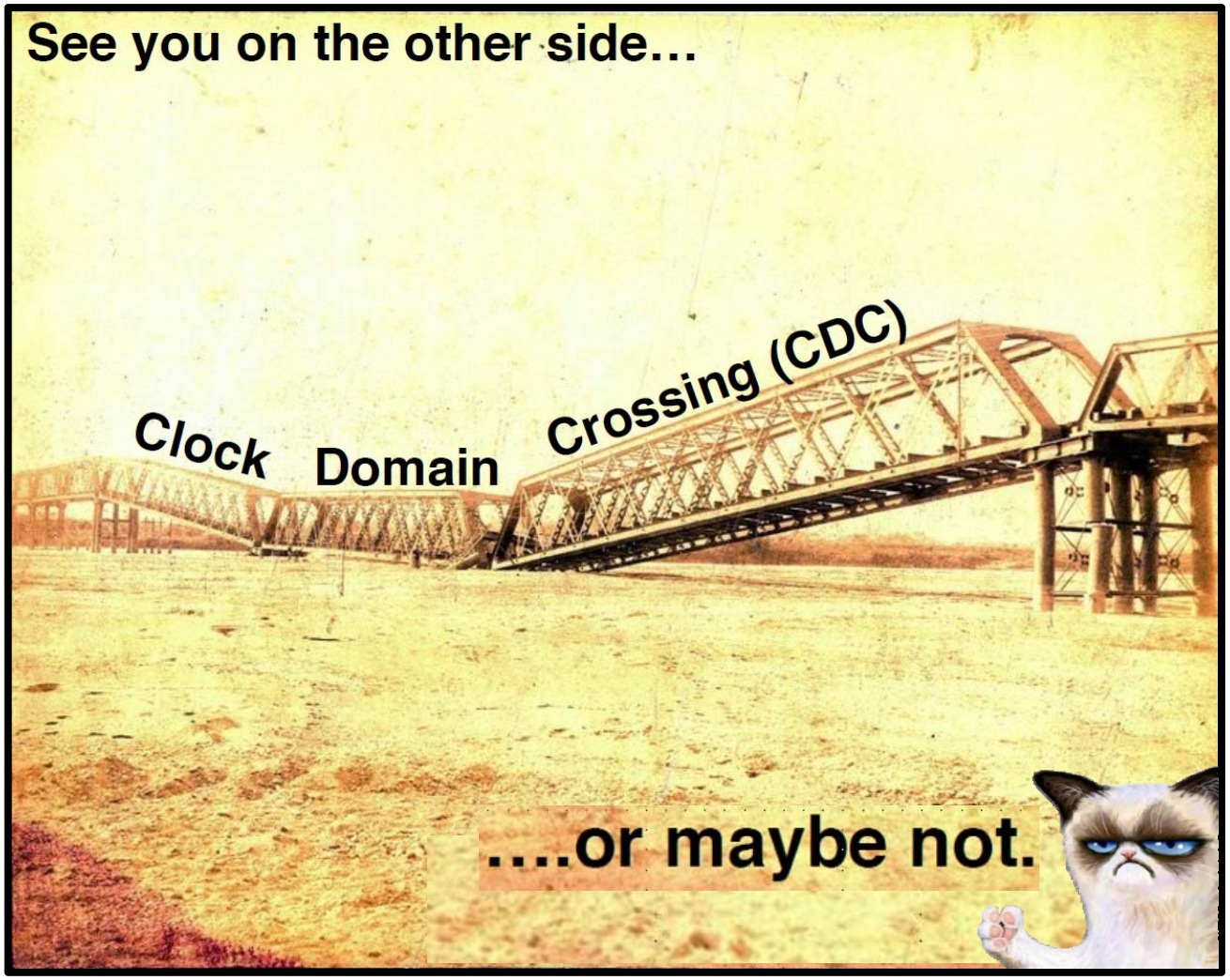
- Clock Domain Crossing (CDC)



FPGA gateware (firmware) design work flow

Design Entry: Timing

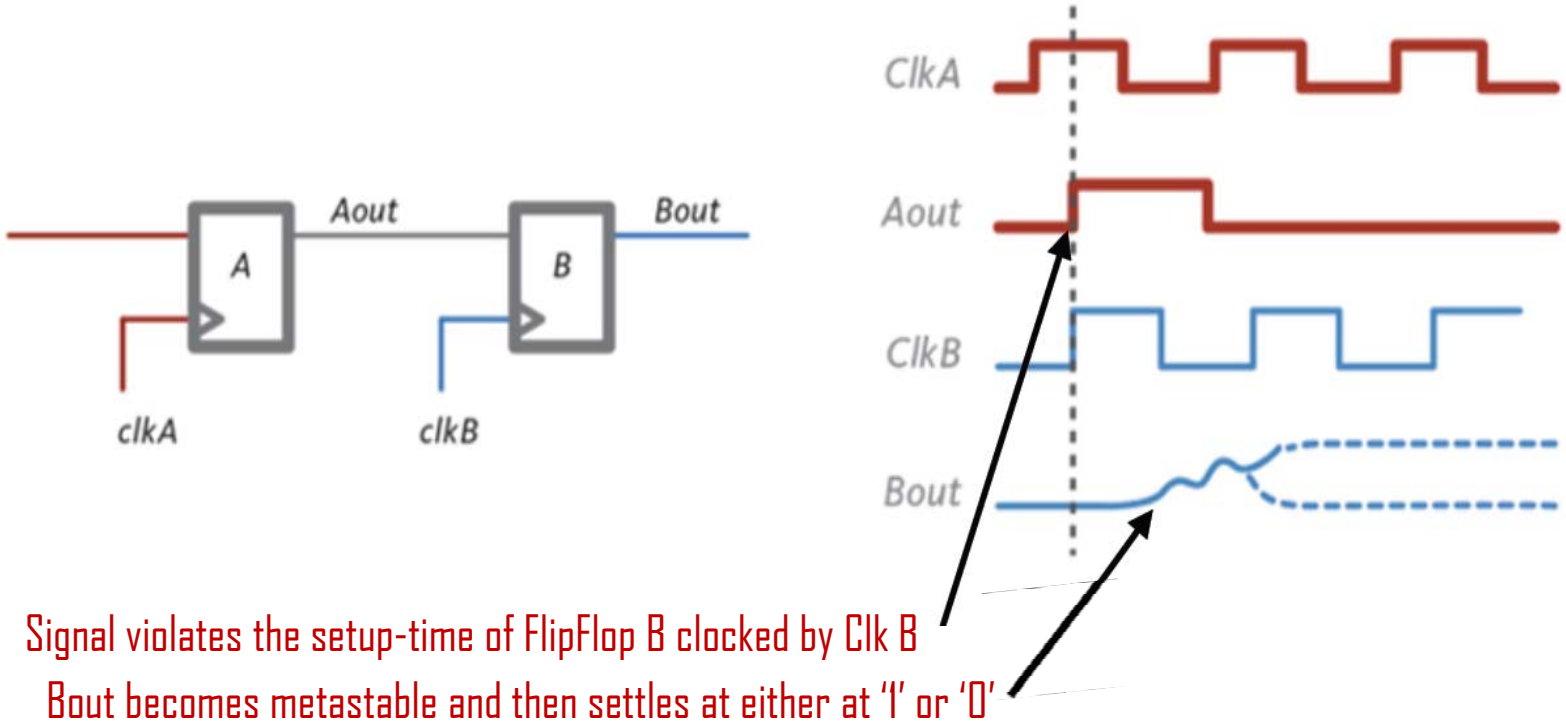
- Clock Domain Crossing (CDC)



FPGA gateware (firmware) design work flow

Design Entry: Timing

- **Clock Domain Crossing (CDC): The problem…**
  - Clock Domain Crossing (CDC) : passing a signal from one clock domain to another (A to B)
  - If clocks are unrelated to each other (asynchronous) timing analysis is not possible
  - Setup and Hold times of FlipFlop B are likely to be violated -> Metastability!!!



Avoid creating unnecessary clock domains

FPGA gateware (firmware) design work flow

Design Entry: Timing

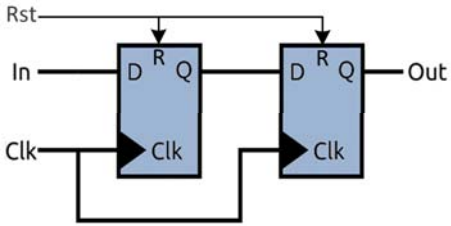
- Clock Domain Crossing: The workaround…

FPGA gateware (firmware) design work flow

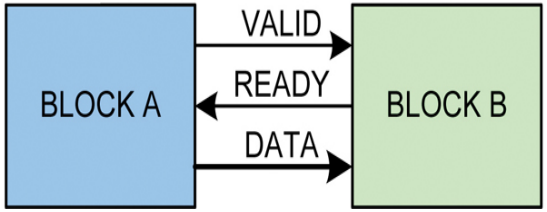
Design Entry: Timing

- **Clock Domain Crossing:** The workaround…
  - Control Paths:

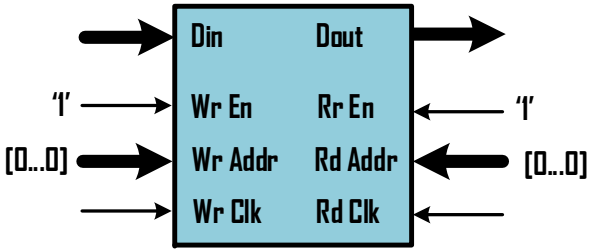
Synchronizers



Handshaking



DPRAM

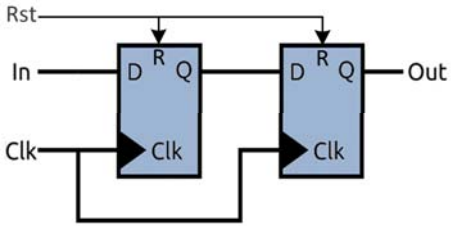


FPGA gateware (firmware) design work flow

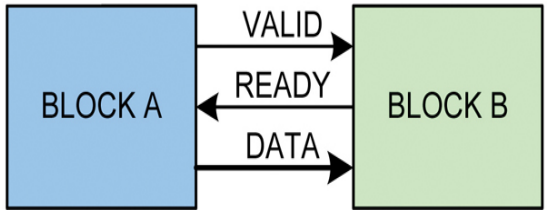
Design Entry: Timing

- **Clock Domain Crossing:** The workaround…
  - Control Paths:

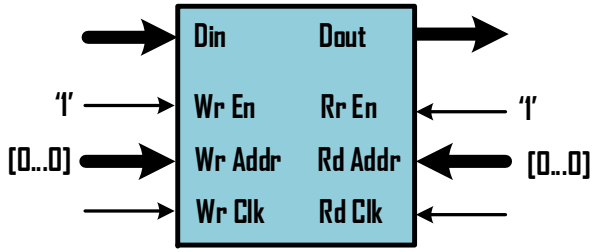
Synchronizers



Handshaking

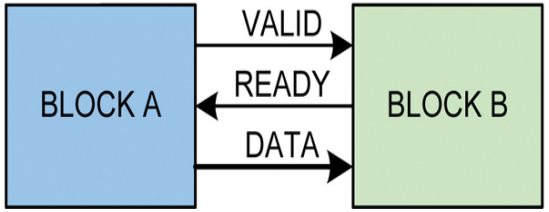


DPRAM



  - Data Paths:

Handshaking



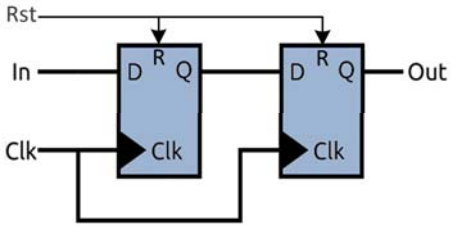
Be aware of FIFO overflow/underflow!!!

FPGA gateware (firmware) design work flow

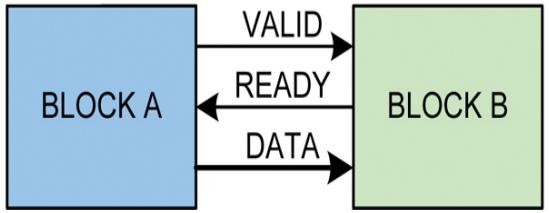
Design Entry: Timing

- **Clock Domain Crossing:** The workaround…
  - Control Paths:

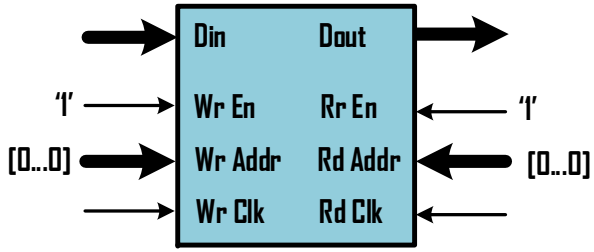
Synchronizers



Handshaking

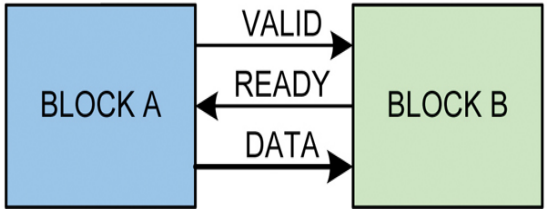


DPRAM



  - Data Paths:

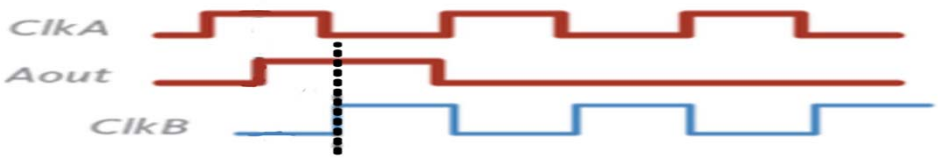
Handshaking



Be aware of FIFO overflow/underflow!!!

  - Control & Data Paths:

Phase alignment



Pros: Low & deterministic latency

Cons: Same clock source, frequency an integer multiple & calibration required

FPGA gateware (firmware) design work flow

Design Entry: Primitives & IP Cores

- **Primitives:** Basic components of the FPGA
  - Vendor (and device) specific
  - Examples: Buffers (I/O & Clock), Registers, BRAMs, DSP blocks, Logic Gates (programed LUTs)
- **Hard IP Cores:** Complex hardware blocks embedded into the FPGA
  - Vendor (and device) specific
  - Fixed I/O location
  - In many cases they may be set through GUI (Wizards)
  - Examples: : PLLs, Multi-gigabit Transceivers, Ethernet MAC, Microprocessors, etc..
- **Soft IP Cores:** Complex (or simple) modules ready to be implemented
  - They may be vendor specific or agnostic:
    - Vendor Specific: Encrypted Code or Requires Hard IP Core
    - Vendor Agnostic: Commercial or Open Source (www.OpenCores.org)
  - In many cases they may be set through GUI (Wizards)
  - Examples: : All kind of modules
- **Two ways of adding Primitives & IP Cores to your system:**
  - Instantiation: The module is EXPLICITELLY added to the system
  - Inference: The module is IMPLICITELLY added to the system

Instantiated FlipFlop
(for Microsemi ProAsic3)

```
DFN1C1 FlipFlop (
    .D   (Input_D),
    .CLK (Clk),
    .CLR (Rst),
    .Q   (Output_Q));
```

Inferred FlipFlop (Verilog)

```
always @(posedge Clk or posedge Rst)
begin
    if (Rst)
        Output_Q <= 0;
    else
        Output_Q <= Input_D;
end
```

FPGA gateware (firmware) design work flow

Design Entry: Primitives & IP Cores

Add Primitives by Inference

- **Primitives:** Basic components of the FPGA
  - Vendor (and device) specific
  - Examples: Buffers (I/O & Clock), Registers, BRAMs, DSP blocks, Logic Gates (programed LUTs)
- **Hard IP Cores:** Complex hardware blocks embedded into the FPGA
  - Vendor (and device) specific
  - Fixed I/O location
  - In many cases they may be set through GUI (Wizards)
  - Examples: : PLLs, Multi-gigabit Transceivers, Ethernet MAC, Microprocessors, etc..
- **Soft IP Cores:** Complex (or simple) modules ready to be implemented
  - They may be vendor specific or agnostic:
    - Vendor Specific: Encrypted Code or Requires Hard IP Core
    - Vendor Agnostic: Commercial or Open Source (www.OpenCores.org)
  - In many cases they may be set through GUI (Wizards)
  - Examples: : All kind of modules
- **Two ways of adding Primitives & IP Cores to your system:**
  - Instantiation: The module is EXPLICITELLY added to the system
  - Inference: The module is IMPLICITELLY added to the system

Instantiated FlipFlop
(for Microsemi ProAsic3)

```
DFN1C1 FlipFlop (
    .D   (Input_D),
    .CLK (Clk),
    .CLR (Rst),
    .Q   (Output_Q));
```

Inferred FlipFlop (Verilog)

```
always @(posedge Clk or posedge Rst)
begin
    if (Rst)
        Output_Q <= 0;
    else
        Output_Q <= Input_D;
end
```

FPGA gateware (firmware) design work flow

Design Entry: Primitives & IP Cores

Add Primitives by Inference

- **Primitives:** Basic components of the FPGA
  - Vendor (and device) specific
  - Examples: Buffers (I/O & Clock), Registers, BRAMs, DSP blocks, Logic Gates (programed LUTs)
- **Hard IP Cores:** Complex hardware blocks embedded into the FPGA
  - Vendor (and device) specific
  - Fixed I/O location
  - In many cases they may be set through GUI (Wizards)
  - Examples: : PLLs, Multi-gigabit Transceivers, Ethernet MAC, Microprocessors, etc..

Add IP Cores by Instantiation (and use the Wizard if possible)

- **Soft IP Cores:** Complex (or simple) modules ready to be implemented
  - They may be vendor specific or agnostic:
    - Vendor Specific: Encrypted Code or Requires Hard IP Core
    - Vendor Agnostic: Commercial or Open Source (www.OpenCores.org)
  - In many cases they may be set through GUI (Wizards)
  - Examples: : All kind of modules
- **Two ways of adding Primitives & IP Cores to your system:**
  - Instantiation: The module is EXPLICITELLY added to the system
  - Inference: The module is IMPLICITELLY added to the system

Instantiated FlipFlop (for Microsemi ProAsic3)

```
DFN1C1 FlipFlop (
    .D    (Input_D),
    .CLK  (Clk),
    .CLR  (Rst),
    .Q    (Output_Q));
```

Inferred FlipFlop (Verilog)

```
always @(posedge Clk or posedge Rst)
begin
    if (Rst)
        Output_Q <= 0;
    else
        Output_Q <= Input_D;
end
```

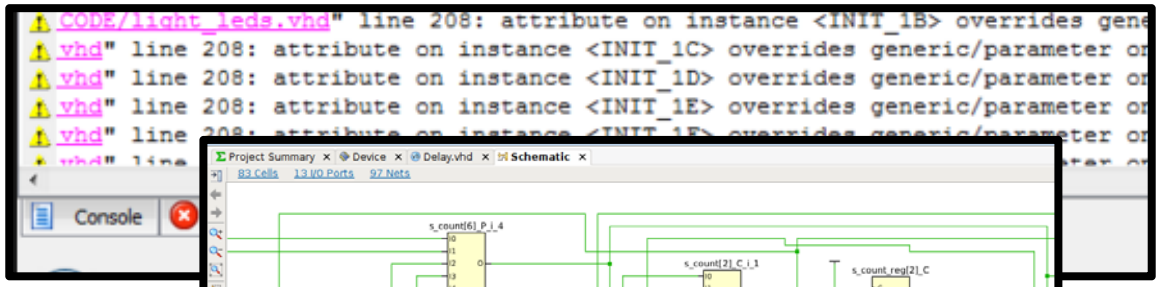
FPGA gateware (firmware) design work flow

Synthesis

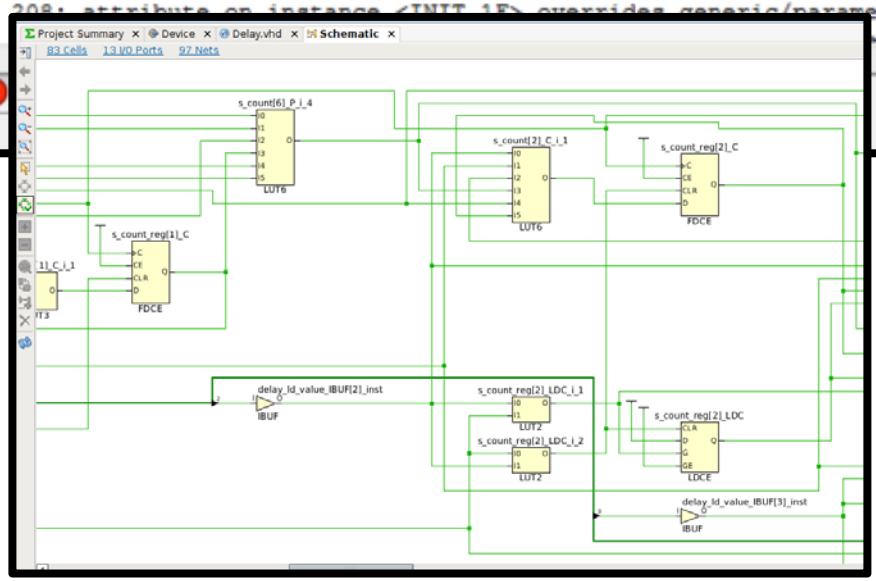
- **What does it do?**
  - Translates the schematic or HDL code into elementary logic functions
  - Defines the connection of these elementary functions
  - Uses Boolean Algebra and Karnaugh maps to optimize logic functions
- **The FPGA design tool optimizes the design during synthesis**

  It may do undesired changes to the system (e.g. remove modules, change signal names, etc.)!!!
- **Always check the synthesis report**
  - Warnings & Errors
  - Estimated resource utilization
  - Optimizations
  - And more…
- **And also check the RTL/Technology viewers**

Example of Synthesis Report



Example of RTL Schematic

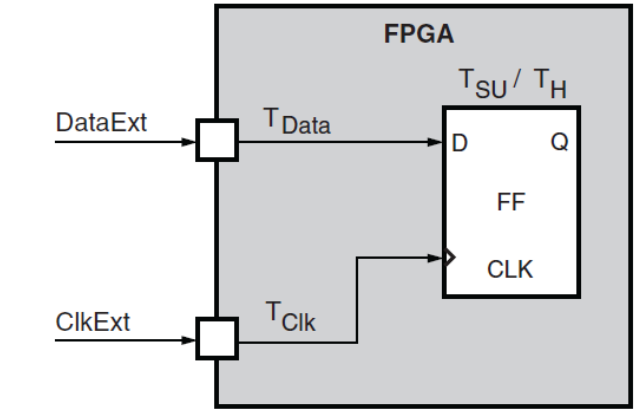


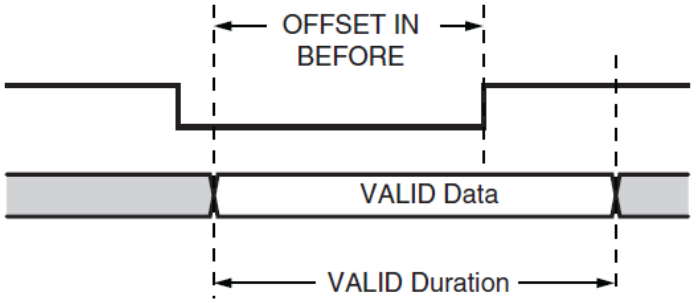
FPGA gateware (firmware) design work flow

Constraints: Timing

- **For a reliable system, the timing requirements for all paths must be provided to the FPGA design tool.**
- **Provided through constraint files (e.g. Xilinx .XDC, etc..) or GUI (that creates/writes constraint files).**
- **The most common types of path categories include:**
  - Input paths
  - Output paths
  - Register-to-register paths (combinatorial paths)
  - Path specific exceptions (e.g. false path, multi-cycle paths, etc.)
- **To efficiently specify these constraints:**
  1) Begin with global constraints (in many cases with this is enough)
  2) Add path specific exceptions as needed
- **Over constraining will difficult the routing**

Example of timing constraint (Xilinx .ucf)



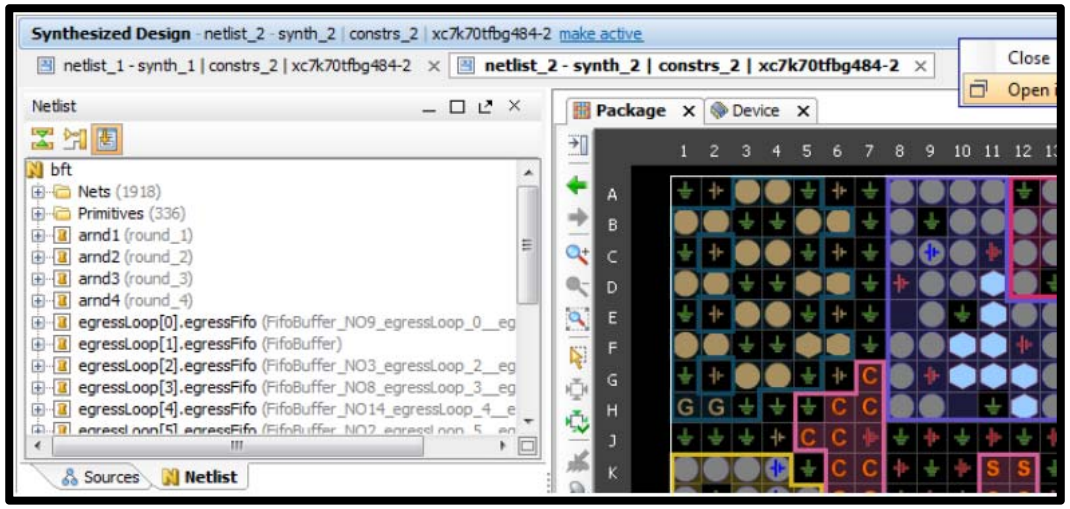



```
TIMEGRP DATA_IN OFFSET = IN 1 VALID 3 BEFORE CLK RISING;
```

FPGA gateware (firmware) design work flow

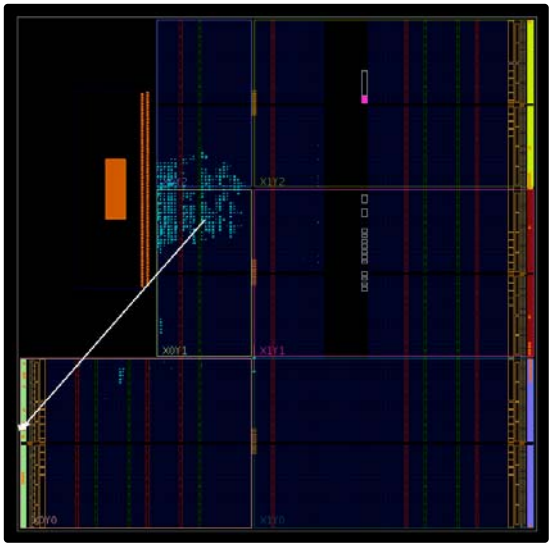
Constraints: Physical

- **Pin planning**



As previously mentioned...
You should do Pin Planning
during Specification Stage

- **Floorplanning**
  - Try to place logic close to their related I/O pins
  - Try to avoid routing across the chip
  - Place the Hard IP cores, the related logic will follow
  - You can separate the logic by areas (e.g. Xilinx Pblocks)

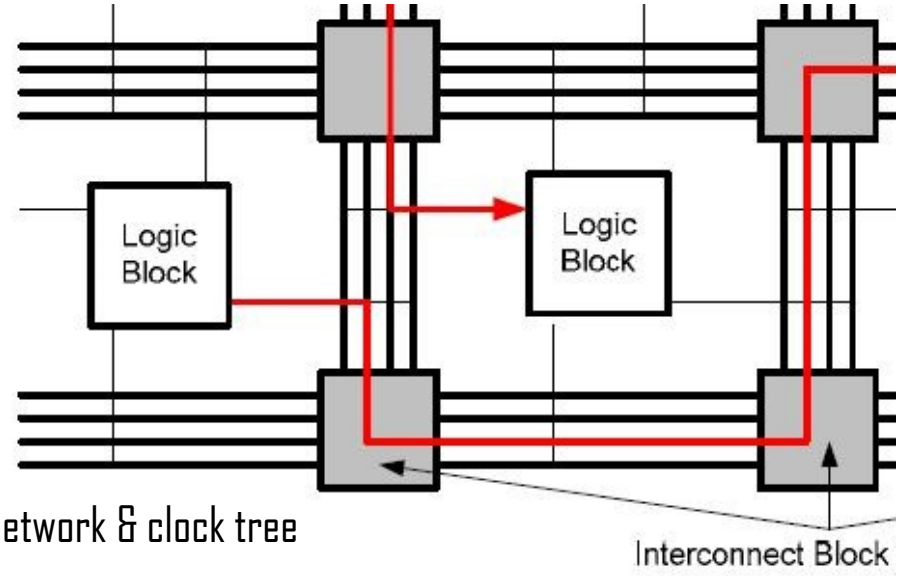


Floorplanning may improve routing times and allow faster system speeds... but too much will difficult the routing!!!

FPGA gateware (firmware) design work flow

Implementation

- **The FPGA design tool:**
  1) Translates the Timing and Physical constraints in order to guide the implementation
  2) Maps the synthesized netlist:
     - Logic elements to FPGA logic cells
     - Hard IP Cores to FPGA hard blocks
     - Verifies that the design can fit the target device
  3) Places and Routes (P&R) the mapped netlist:
     - Physical placement of the FPGA logic cells
     - Physical placement of the FPGA hard blocks
     - Routing of the signals through the interconnect network & clock tree



- **The FPGA design tool may be set for different optimizations (Speed, Area, Power or default)**
- **Physical Placement & Timing change after re-implementing (use constraints to minimize these changes)**
- **You should always check the different reports generated during implementation**

FPGA gateware (firmware) design work flow

Static Timing Analysis

- **The FPGA design tool analyses the signals propagation delays and clock relationships after P&R**
- **A timing report is generated, including the paths that did not meet the timing requirements**
- **Rule of thumb for timing violations:**

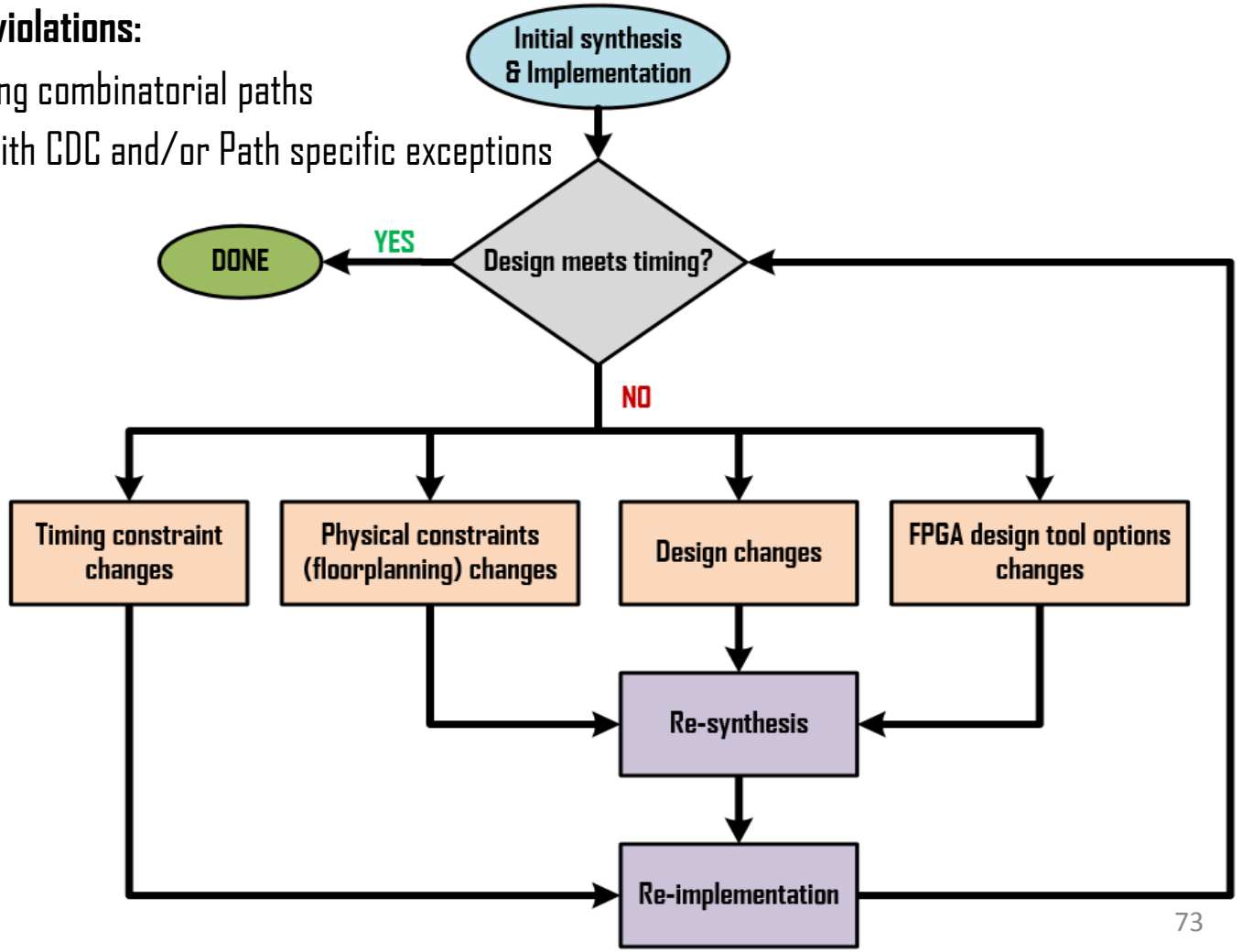
  - Setup violations: Too long combinatorial paths
  - Hold violations: Issue with CDC and/or Path specific exceptions
- **The timing closure flow:**

FPGA gateware (firmware) design work flow

Static Timing Analysis

- The FPGA design tool analyses the signals propagation delays and clock relationships after P&R
- A timing report is generated, including the paths that did not meet the timing requirements
- Rule of thumb for timing violations:

  - Setup violations: Too long combinatorial paths
  - Hold violations: Issue with CDC and/or Path specific exceptions
- The timing closure flow:



FPGA gateware (firmware) design work flow

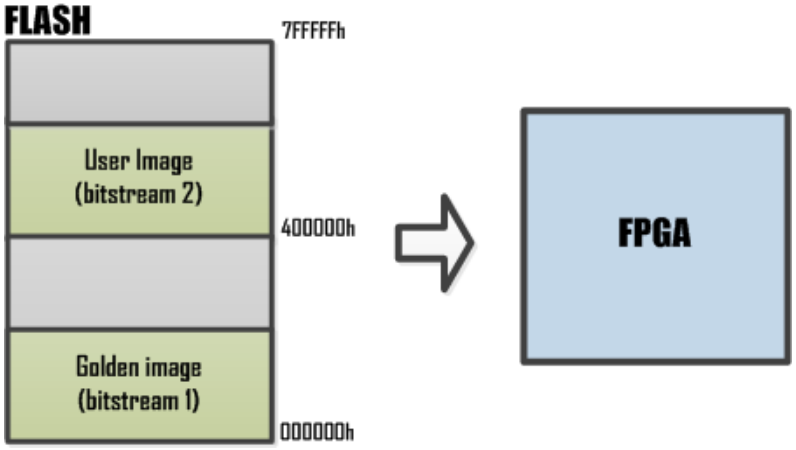
Bitstream Generation & FPGA Programming

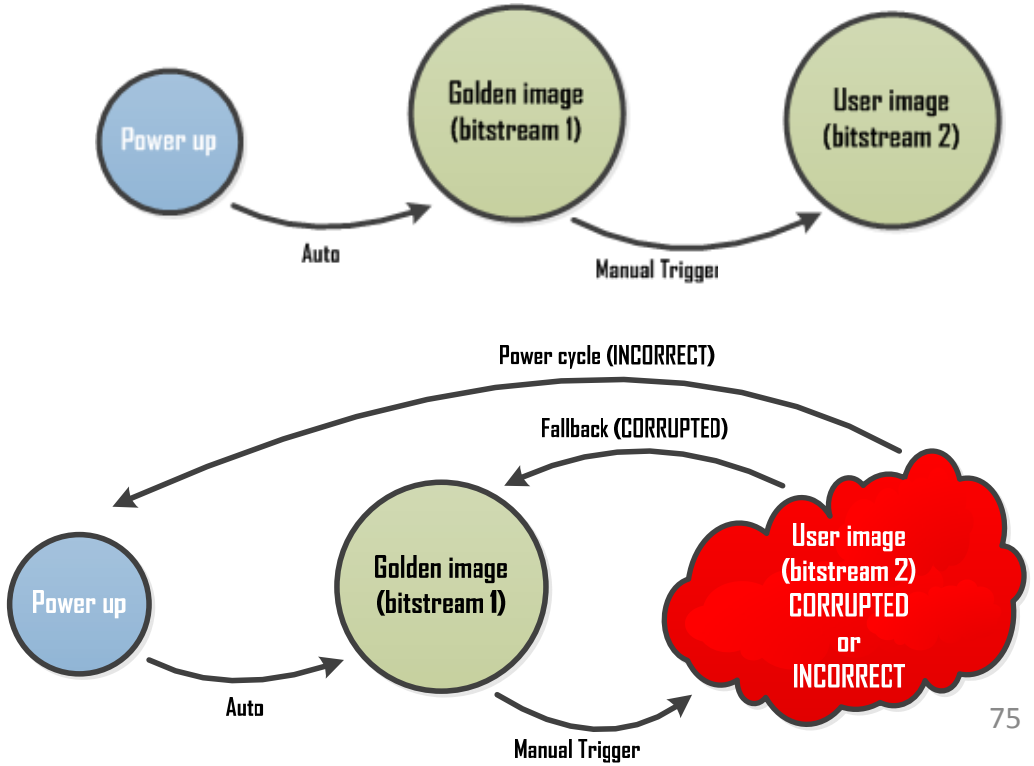
- **Bitstream:**
  - Binary file containing the FPGA configuration data
  - Each FPGA vendor has its own bitstream file extension (e.g. .bit (Xilinx), .sof (Altera))
- **FPGA programming:**
  - Bitstream is loaded into the FPGA through JTAG
  - Configuration data may be stored in on-board FLASH and loaded by the FPGA at power up
  - Multiboot/Safe FPGA configuration

FPGA gateware (firmware) design work flow

Bitstream Generation & FPGA Programming

- **Bitstream:**
  - Binary file containing the FPGA configuration data
  - Each FPGA vendor has its own bitstream file extension (e.g. .bit (Xilinx), .sof (Altera))
- **FPGA programming:**
  - Bitstream is loaded into the FPGA through JTAG
  - Configuration data may be stored in on-board FLASH and loaded by the FPGA at power up
  - Multiboot/Safe FPGA configuration



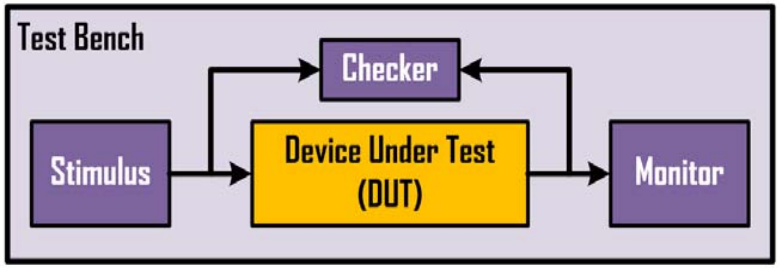


Multiboot/Safe FPGA configuration diagrams

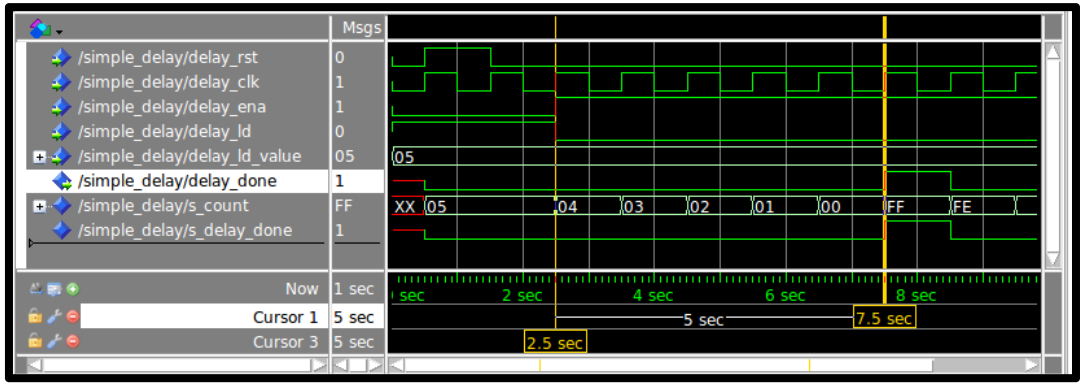
FPGA gateware (firmware) design work flow

Simulation

- **Event-based simulation to recreate the parallel nature of digital designs**
- **Verification of HDL modules and/or full systems**
- **HDL simulators:**
  - Most popular: Modelsim
  - Other simulators: Vivado Simulator (Xilinx), Icarus Verilog (Open-source), etc..
- **Different levels of simulation**
  - Behavioural: simulates only the behaviour of the design — **Fast**
  - Functional: uses realistic functional models for the target technology — **Slow**
  - Timing : most accurate. Uses Implemented design after timing analysis — **Very Slow**



Example of simulator wave window



FPGA gateware (firmware) design work flow

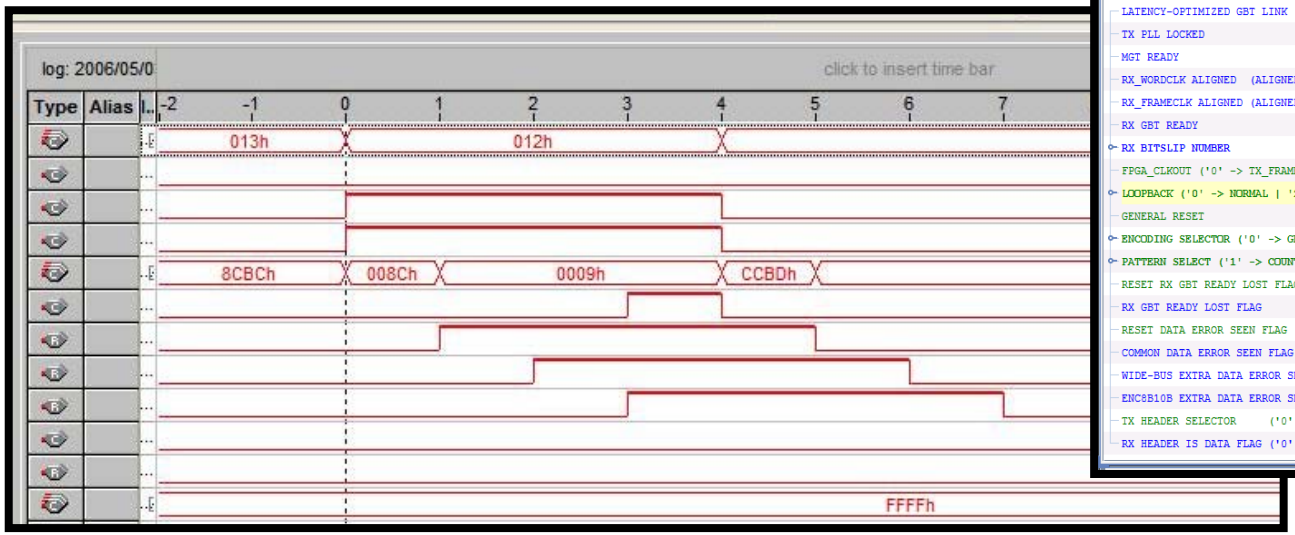
In-System Analysers & Virtual I/Os

- Your design is up… and also running?
- Most FPGA vendors provide in-system analyzers & virtual I/Os
- Can be embedded into the design and controlled by JTAG
- Allow monitoring but also control of the FPGA signals
- Minimize interfering with the your system by:

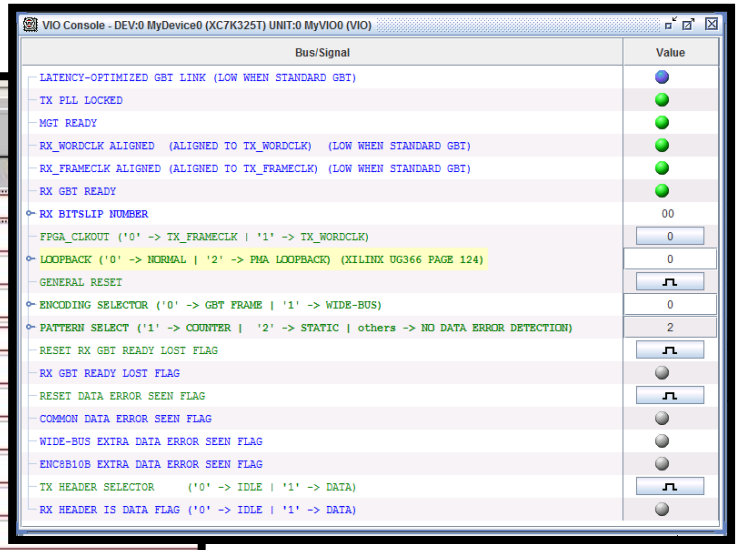
Placing extra registers between the monitored signals and the In-System Analyser

- It is useful to spy inside the FPGA… but the issue may come form the rest of the board!!!
- Remember… it is HARDWARE

Example of In-System Analyser (Altera SignalTap II)



Example of Virtual I/Os (Xilinx VIO)



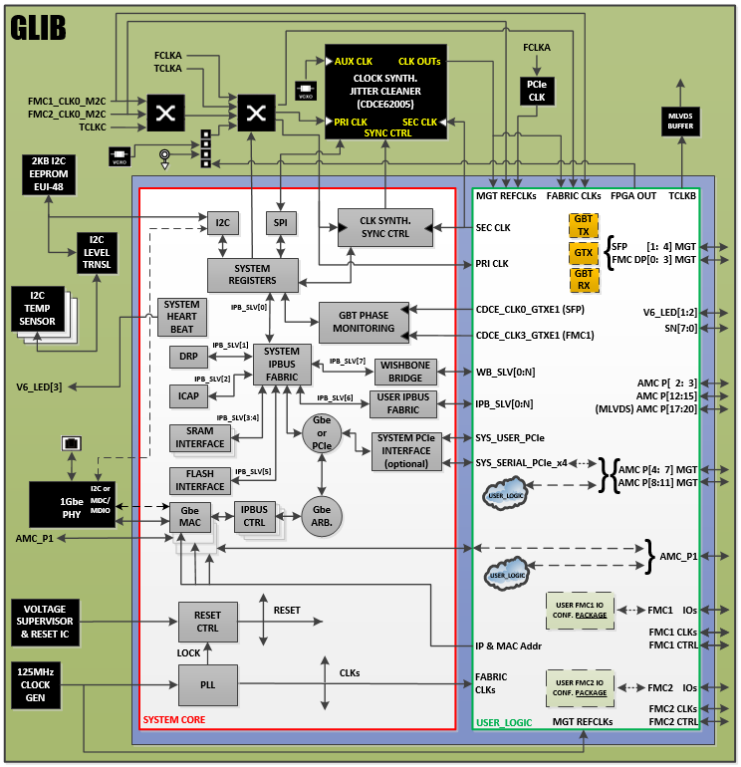
FPGA gateware (firmware) design work flow

Debugging Techniques

FPGA gateware (firmware) design work flow

Debugging Techniques

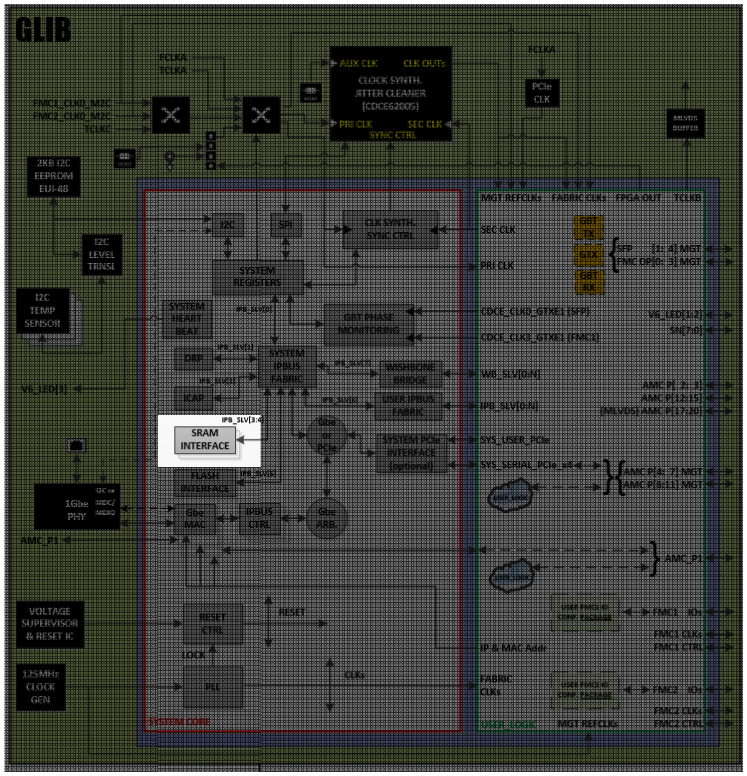
Divide & Conquer



FPGA gateware (firmware) design work flow

Debugging Techniques

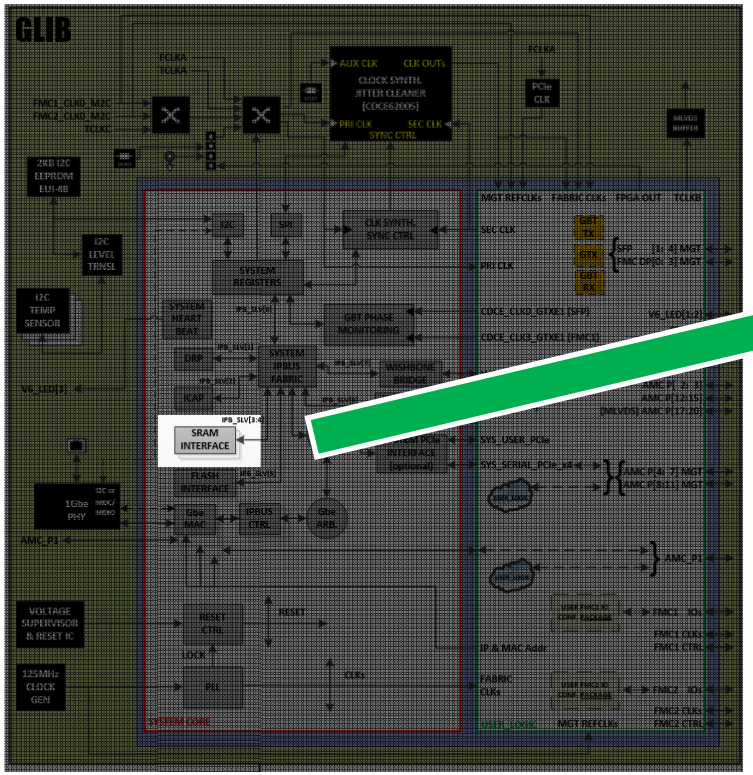
Divide & Conquer

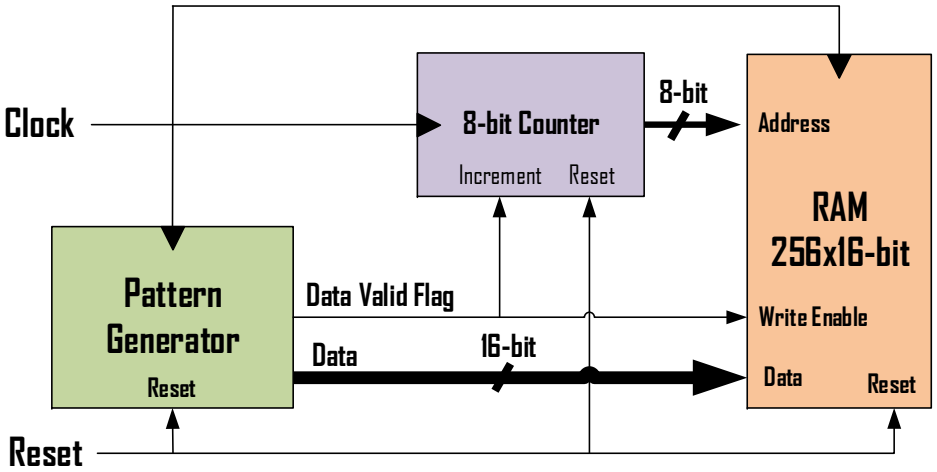


FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer

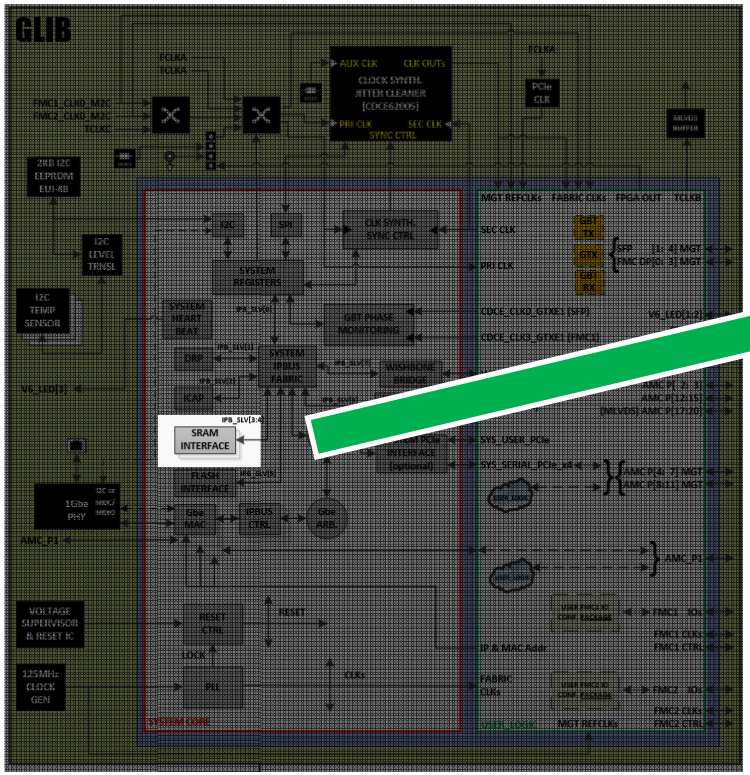




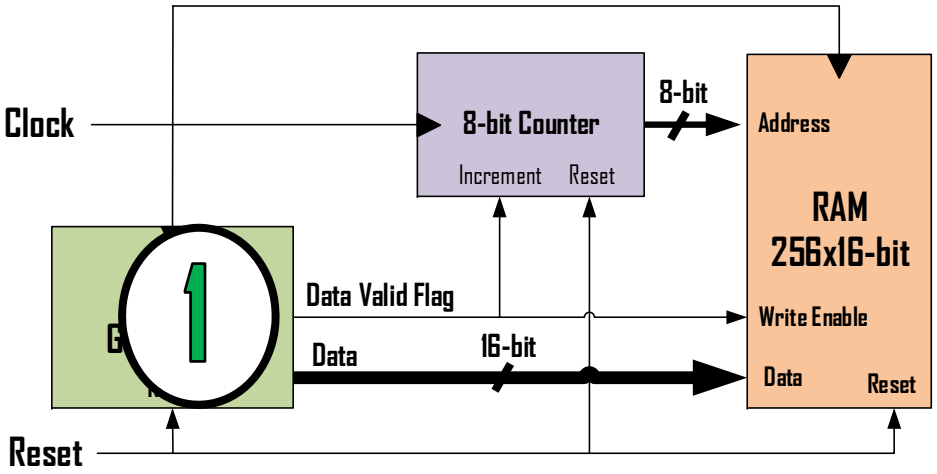
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



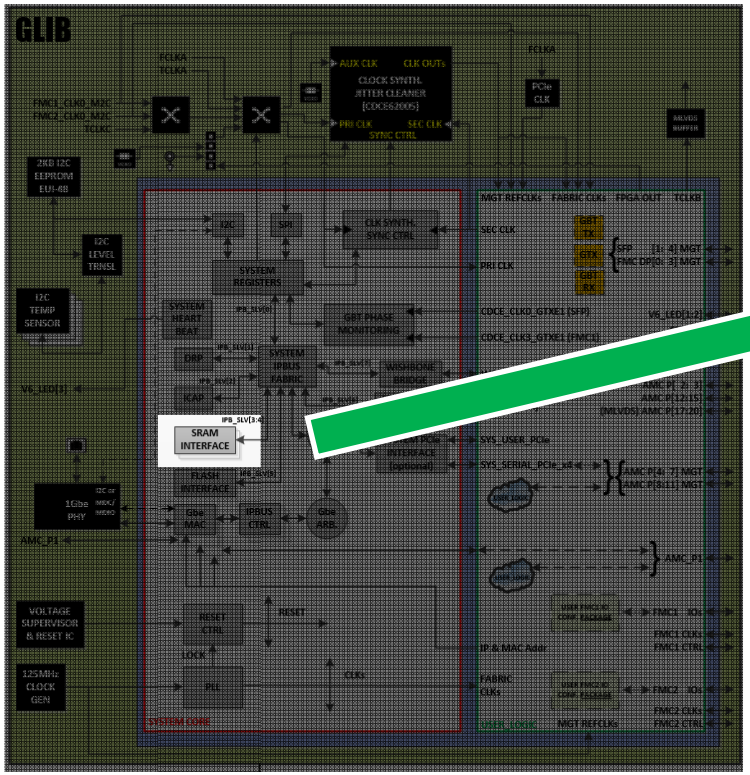
Follow the chain



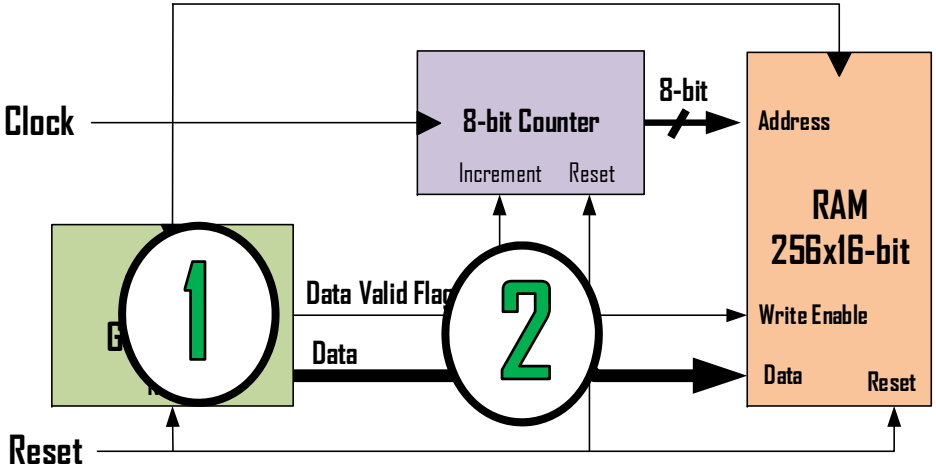
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



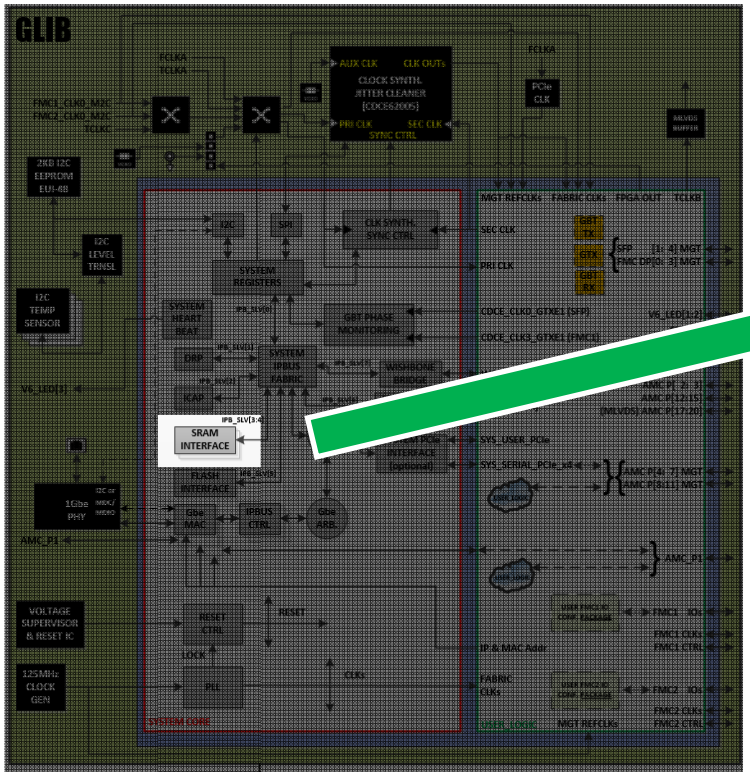
Follow the chain



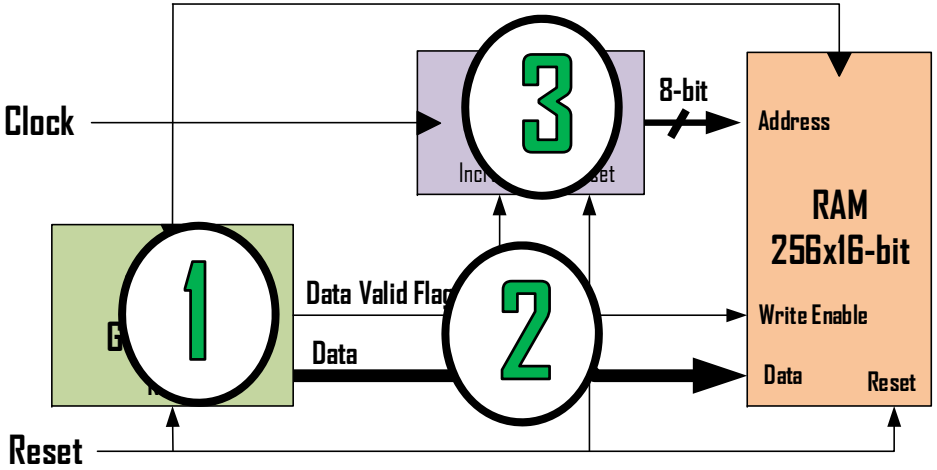
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



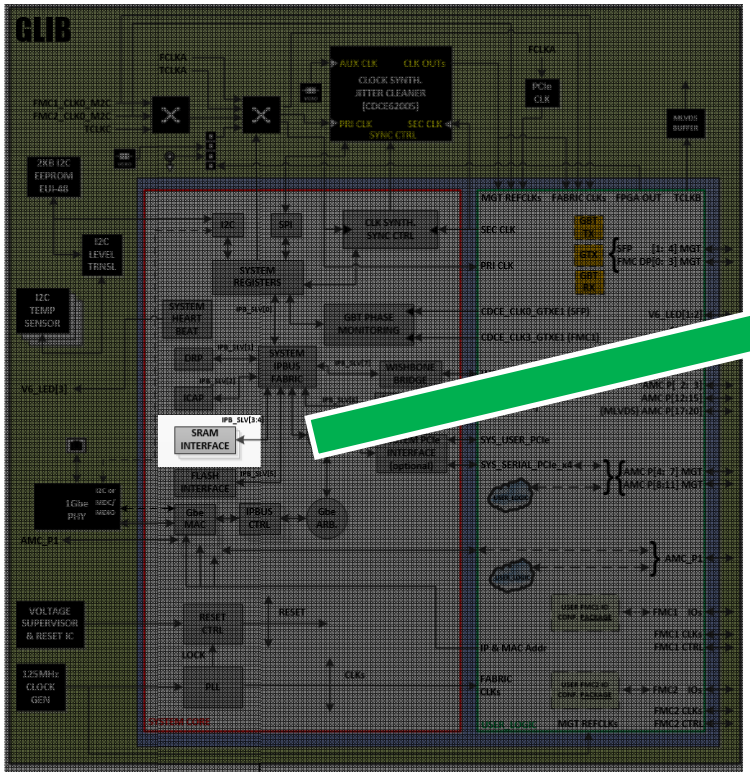
Follow the chain



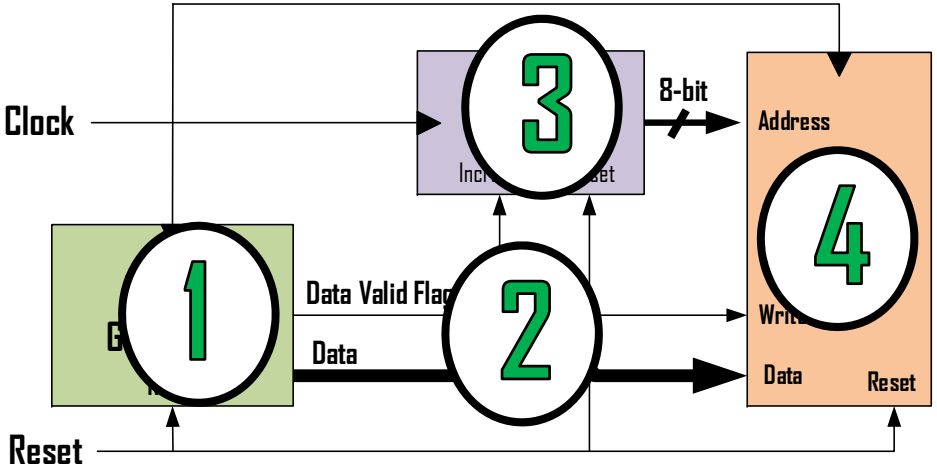
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



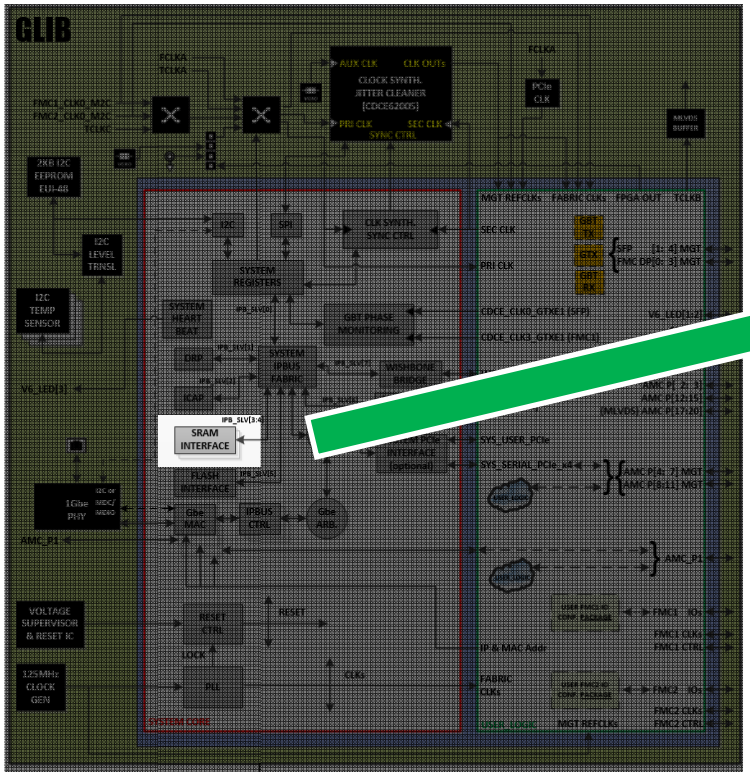
Follow the chain



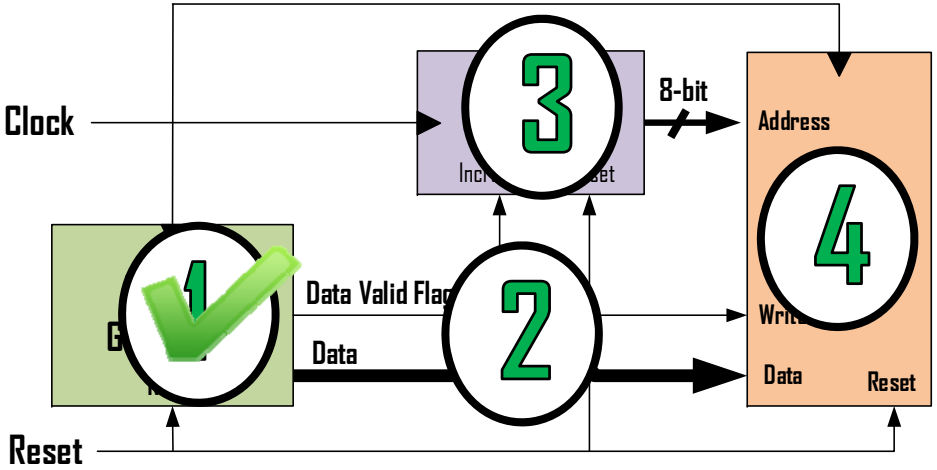
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



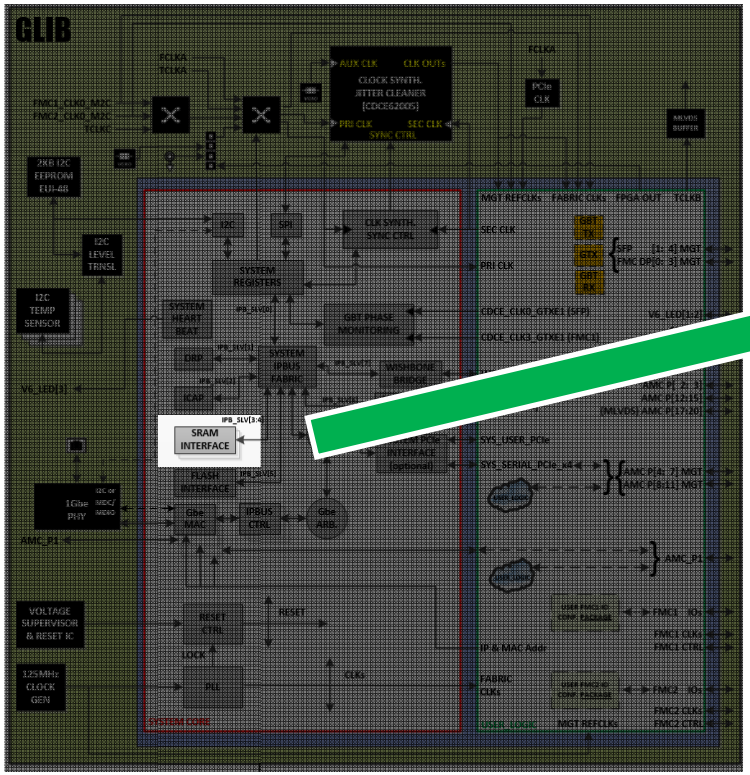
Follow the chain



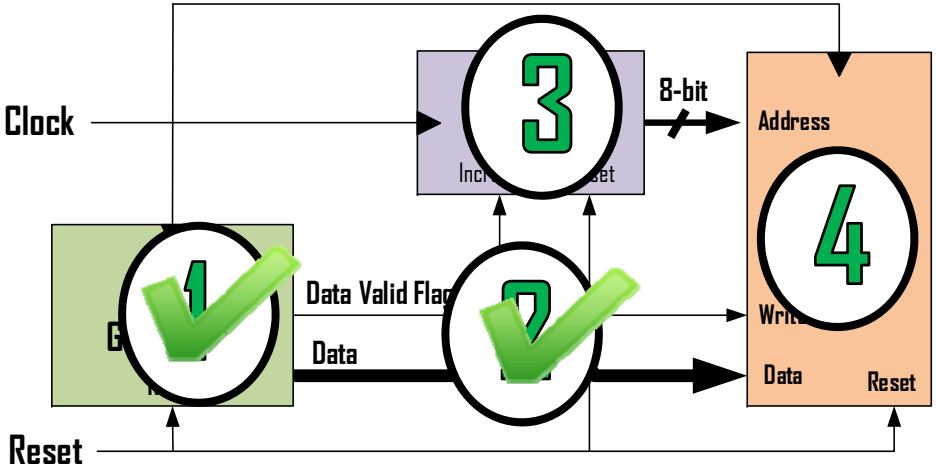
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



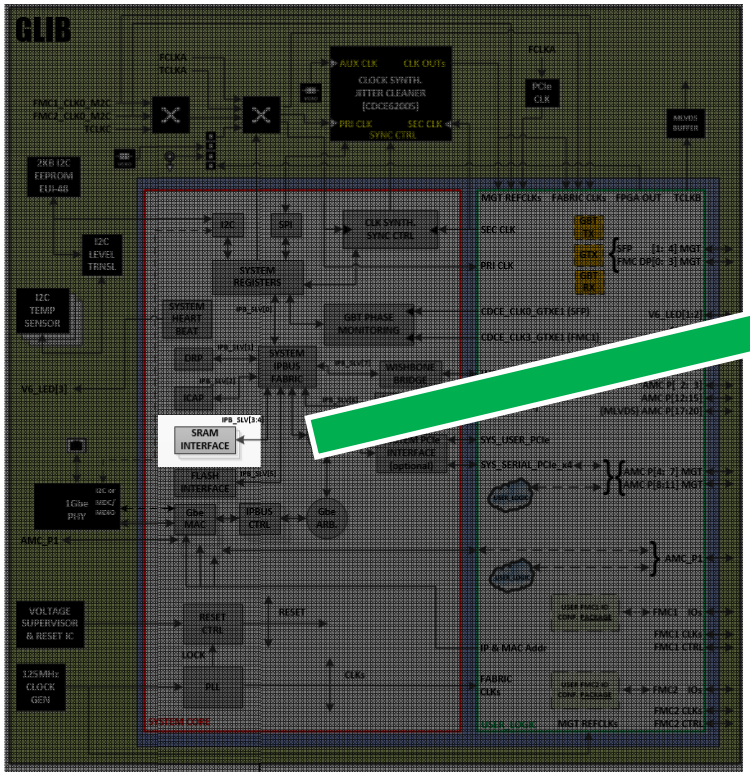
Follow the chain



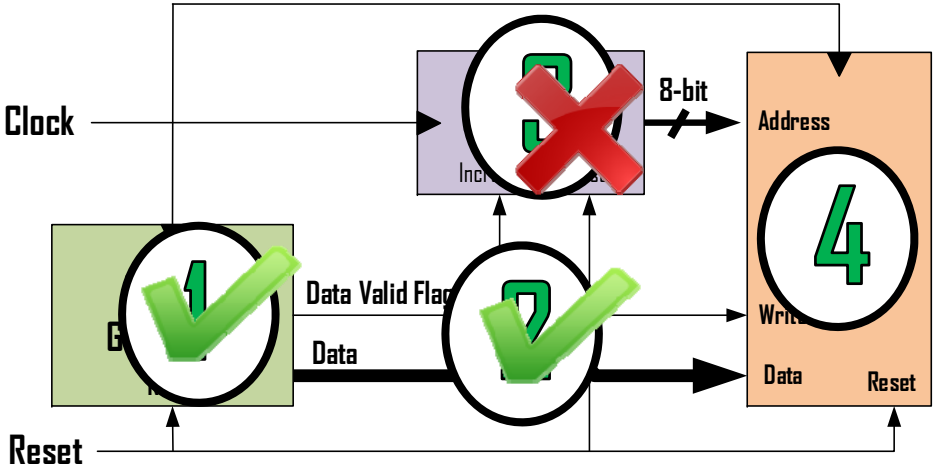
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



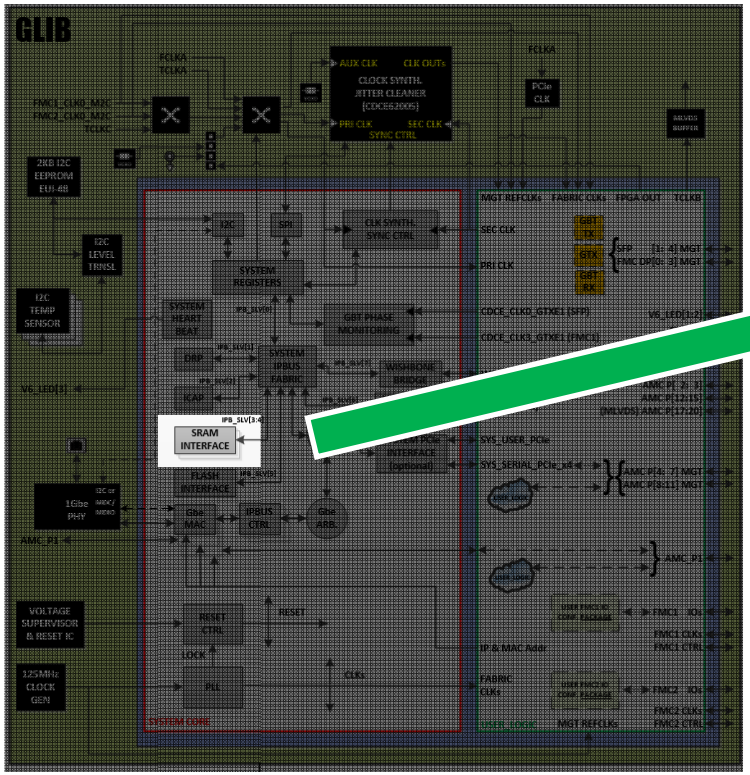
Follow the chain



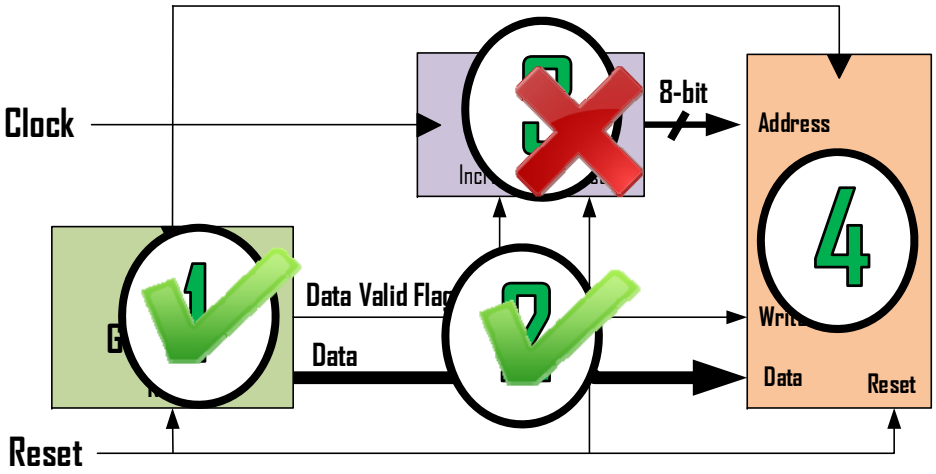
FPGA gateware (firmware) design work flow

Debugging Techniques

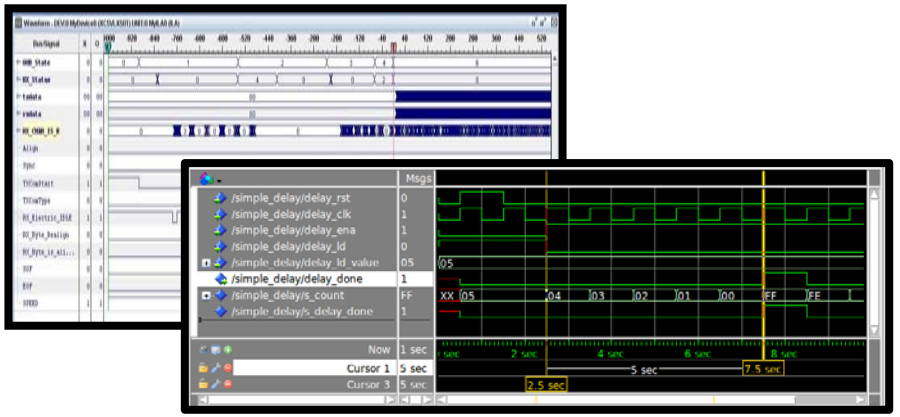
Divide & Conquer



Follow the chain



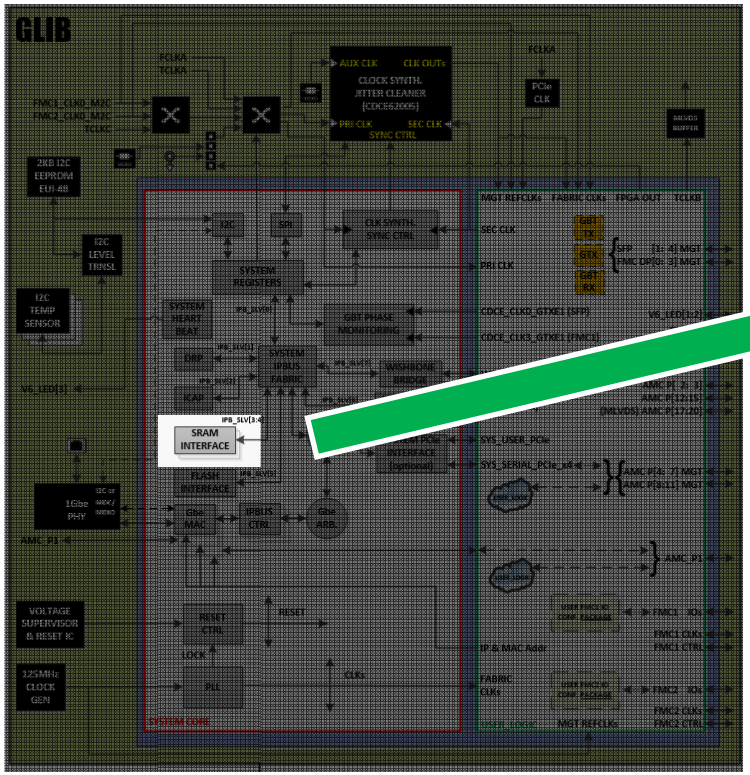
Open the box



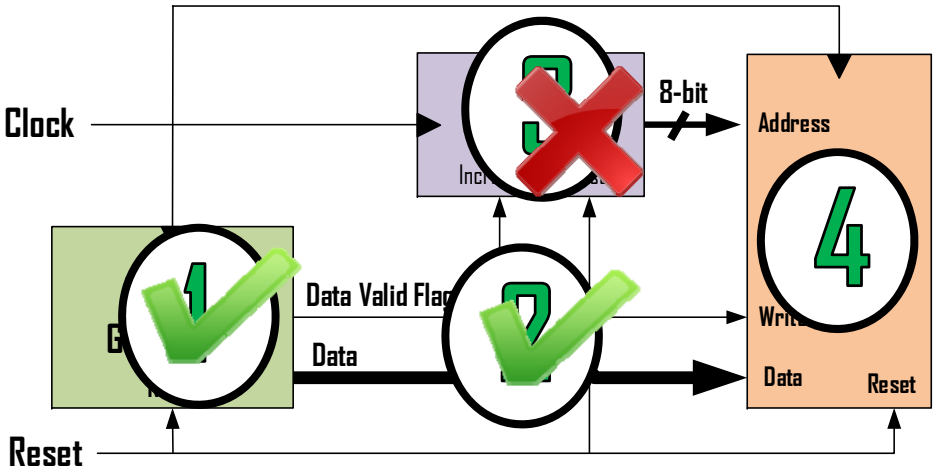
FPGA gateware (firmware) design work flow

Debugging Techniques

Divide & Conquer



Follow the chain



Open the box

We are debugging HARDWARE!!!

FPGA gateware (firmware) design work flow

After debugging…

FPGA gateware (firmware) design work flow

After debugging…

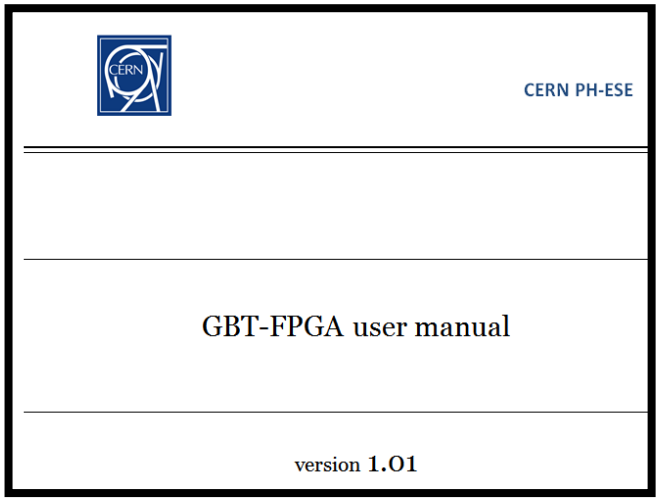
FPGA gateware (firmware) design work flow

After debugging…

FPGA gateware (firmware) design work flow

After debugging…

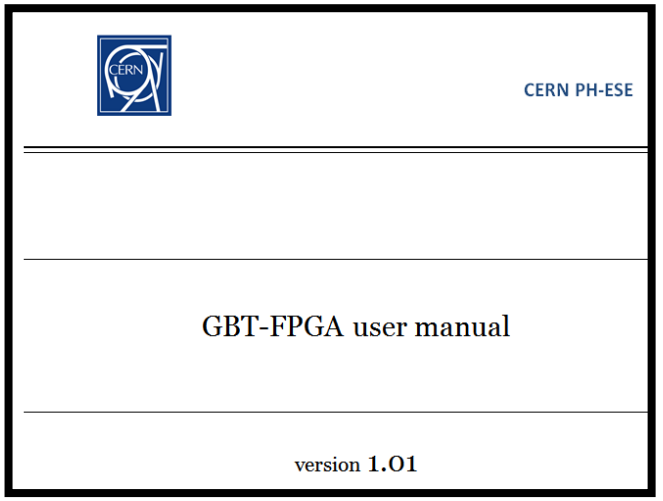
- Documentation



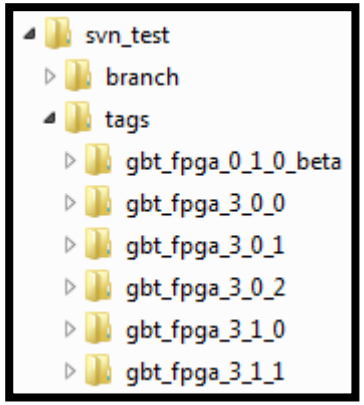
FPGA gateware (firmware) design work flow

After debugging…

- Documentation



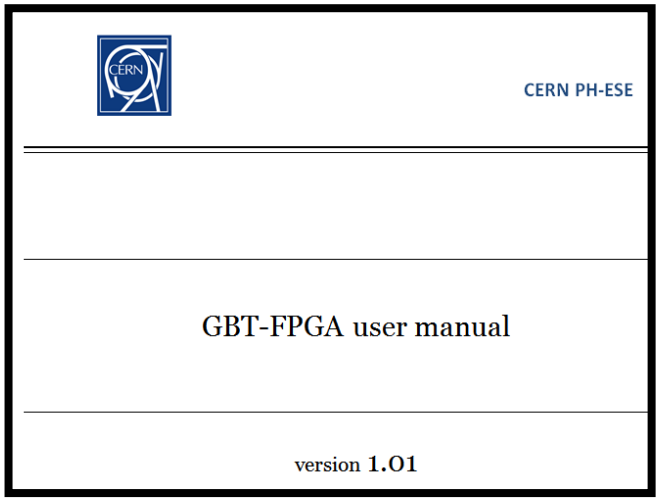
- Maintenance



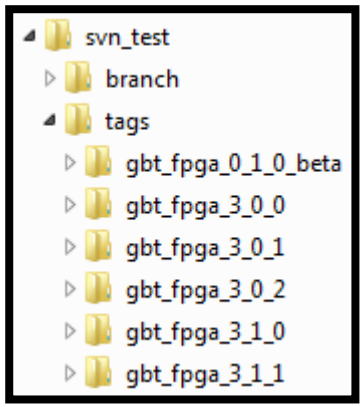
FPGA gateware (firmware) design work flow

After debugging…

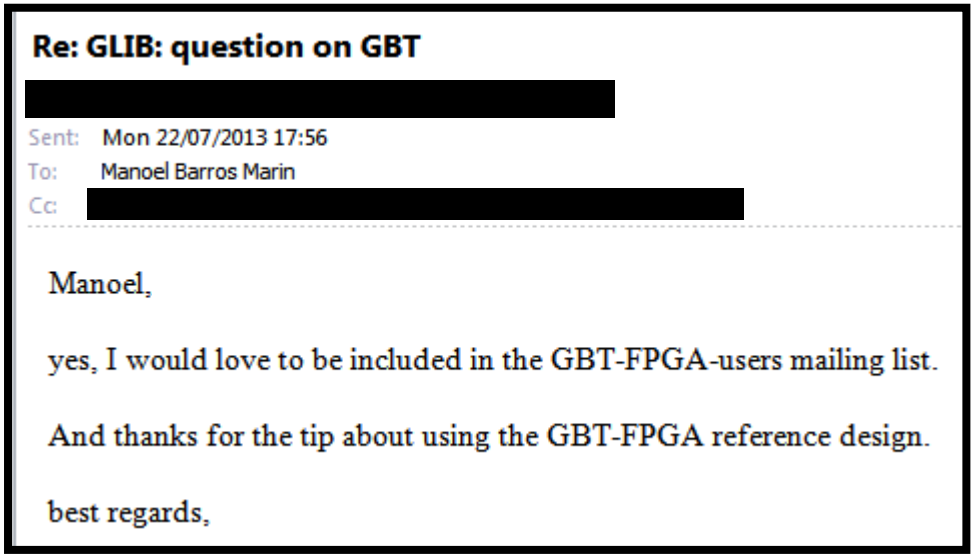
- Documentation



- Maintenance



- … and maybe User Support



Advanced FPGA design

ISOTDAQ 2016 @ Rehovot (Israel)

29/01/2016

Outline:

- ...from the previous lesson
- Key concepts about HDL
- FPGA gateware (firmware) design work flow
- **Summary**

Manoel Barros Marin

BE-BI-QP

Summary

- **FPGA - Wikipedia**

  A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

- **Key concepts about FPGA design**
  - FPGA gateware (firmware) design is NOT programming
  - HDL are used for describing HARDWARE
  - Timing in FPGA gateware (firmware) design is critical
- **FPGA gateware (firmware) design flow**
  - Plan, plan and plan again
  - Modular and reusable system
  - Coding for synthesis
  - Take care of your resets and clocks schemes
  - Clock Domain Crossing is tricky
  - You must properly constraint your design
  - Optimize in your code but also with constraints and FPGA design tool options
  - Read the reports (Synthesis, Implementation & Static Timing Analysis)
  - Try to be methodic when debugging & use all tools available
  - A running system is not the end of the road… (Documentation, Maintenance. User Support)

Summary

- **FPGA - Wikipedia**

  A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

- **Key concepts about FPGA design**
  - FPGA gateware (firmware) design is NOT programming
  - HDL are used for describing HARDWARE
  - Timing in FPGA gateware (firmware) design is critical
- **FPGA gateware (firmware) design flow**
  - Plan, plan and plan again
  - Modular and reusable system
  - Coding for synthesis
  - Take care of your resets and clocks schemes
  - Clock Domain Crossing is tricky
  - You must properly constraint your design
  - Optimize in your code but also with constraints and FPGA design tool options
  - Read the reports (Synthesis, Implementation & Static Timing Analysis)
  - Try to be methodic when debugging & use all tools available
  - A running system is not the end of the road… (Documentation, Maintenance. User Support)

Summary

- **FPGA - Wikipedia**

  A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

- **Key concepts about FPGA design**
  - FPGA gateware (firmware) design is NOT programming
  - HDL are used for describing HARDWARE
  - Timing in FPGA gateware (firmware) design is critical
- **FPGA gateware (firmware) design flow**
  - Plan, plan and plan again
  - Modular and reusable system
  - Coding for synthesis
  - Take care of your resets and clocks schemes
  - Clock Domain Crossing is tricky
  - You must properly constraint your design
  - Optimize in your code but also with constraints and FPGA design tool options
  - Read the reports (Synthesis, Implementation & Static Timing Analysis)
  - Try to be methodic when debugging & use all tools available
  - A running system is not the end of the road… (Documentation, Maintenance. User Support)

But it works ☺

Summary

- **FPGA - Wikipedia**

  A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable".

- **Key concepts about FPGA design**
  - FPGA gateware (firmware) design is NOT programming
  - HDL are used for describing HARDWARE
  - Timing in FPGA gateware (firmware) design is critical
- **FPGA gateware (firmware) design flow**
  - Plan, plan and plan again
  - Modular and reusable system
  - Coding for synthesis
  - Take care of your resets and clocks schemes
  - Clock Domain Crossing is tricky
  - You must properly constraint your design
  - Optimize in your code but also with constraints and FPGA design tool options
  - Read the reports (Synthesis, Implementation & Static Timing Analysis)
  - Try to be methodic when debugging & use all tools available
  - A running system is not the end of the road… (Documentation, Maintenance. User Support)

Where do I find more info about this??

There are nice papers & books but… FPGA vendors provide very good documentation about all topics mentioned in this lecture

But it works ☺

Acknowledges

- Markus Joos (CERN) & organisers of ISOTDAQ-16
- Andrea Borga (NIKHEF), Torsten Alt (FIAS) for their contribution to this lecture
- Rhodri Jones, Thibaut Lefevre, Andrea Boccardi & other colleagues from CERN BE-BI-QP

Any Questions?